Using Remote Attestation of Trust for Computer Forensics

Gabriela Claret Limonta Marquez

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 20.11.2018

Supervisor

Prof. Raimo Kantola

Advisor

Dr. Ian Oliver

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Gabriela Claret Limonta Marquez

Title Using Remote Attestation of Trust for Computer Forensics

Degree programme Computer, Communication and Information Sciences

Major Communications Engineering **Code of major** ELEC3029

Supervisor Prof. Raimo Kantola

Advisor Dr. Ian Oliver

Date 20.11.2018 **Number of pages** 94+34 **Language** English

Abstract

Telecommunications systems are critical systems with high quality of service constraints. In Network Function Virtualization (NFV), commonly known as the Telco Cloud, network functions are distributed as virtual machines that run on generic servers in a datacenter. These network functions control critical elements; therefore, they should be run on trusted hardware.

Trusted computing concepts can be used to guarantee the trustworthiness of the underlying hardware platform running critical workload. These concepts include the Trusted Platform Module and Remote Attestation. This work identifies limitations in existing solutions and uses those as motivation for designing and implementing a finer-grained definition of trust.

This thesis designs and develops a remote attestation solution, which includes a policy and rule based mechanism for determining platform trust in a trusted cloud. Additionally, it develops a fine-grained concept of trust in a cloud environment based on NFV. Finally, this thesis utilizes the remote attestation solution to develop a forensics system based on root cause analysis, which allows the investigation of attestation failures and their mitigation.

Keywords Trusted Computing, NFV, TPM, Cloud Computing, Telecommunications, RCA

Preface

I would like to express my most sincere gratitude to my advisor, Dr. Ian Oliver, for taking me on as his student, for his constant support and encouragement during the time we have been working together, for his patience and guidance when I came across obstacles and for motivating me to pursue new challenges. Thank you for the countless hours of mentoring and for making research a very fun and exciting experience.

I would like to thank Prof. Raimo Kantola for his assistance during this project, for the valuable suggestions and insights he provided when discussing my work and for giving me the freedom and trust to work on my thesis independently.

Special thanks to Nokia Networks for funding my thesis work and to my managers during this time Markku Niiranen and Martin Peylo.

I also had great pleasure of working with the Cybersecurity Research Team at Nokia Bell Labs, and I would like to thank them for all the wonderful coffee time discussions. I would like to thank the team members: Dr. Yoan Miche, Dr. Silke Holtmanns, Aapo Kalliola and Leo Hippeläinen for their support and for sharing their expertise and helping me grow as a researcher. I am particularly grateful for the assistance given by Dr. Yoan Miche, who provided valuable comments when proofreading earlier drafts of this work. Also, I would like to thank my fellow student team members: Borger, Sakshyam and Isha for their support and friendship.

I would like to thank my family and friends for their constant support. Thanks to my parents, Gaudy and Eugenio, who have supported me every step of the way, as well as my brother Santiago. Thanks also to my friends Vane and Magy, who always had time to listen to me talk about this work.

Finally, I want to express my deepest appreciation to Antti for his unconditional support, encouragement and love, without you none of this would have been possible, thank you so much.

Otaniemi, November 2018

Gabriela Claret Limonta Marquez

Contents

Abbreviations

ABRMD	Access Broker and Resource Management Daemon
ACM	Authenticated Code Module
AK	Attestation Key
API	Application Programming Interface
ARM	Advanced RISC Machine
BIOS	Basic Input/Output System
BSS	Business Support Systems
CFT	Causal Factor Tree
CRTM	Core Root of Trust Measurement
DRTM	Dynamic Root of Trust Measuremen
EK	Endorsement Key
EM	Element Management
EPC	Evolved Packet Core
ESAPI	Enhanced System Application Programming Interface
ETSI	European Telecommunications Standards Institute
GPT	GUID Partition Table
GRUB	GRand Unified Bootloader
GUID	Globally Unique Identifier
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IMA	Integrity Measurement Architecture
IoT	Internet of Things
JSON	Javascript Object Notation
LSM	Linux Security Module
MANO	Management and Operations
MLE	Measured Launch Environment
MME	Mobility Management Entity
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NUC	Next Unit of Computing
NV	Non-Volatile
NVRAM	Non-Volatile Random Access Memory
OS	Operating System
OSS	Operations Support Systems
OSS/BSS	Operations and Business Support Systems
OpenCIT	Open Cloud Integrity Technology
PCR	Platform Configuration Register
PMS	Patch Management System
RCA	Root Cause Analysis
REST	Representational State Transfer
RISC	Reduced Instruction Set Computer
RNG	Random Number Generator
ROM	Read-Only Memory

RSA	Rivest–Shamir–Adleman
SAPI	System Application Programming Interface
SDN	Software Defined Networking
SELinux	Security Enhanced Linux
SHA	Secure Hash Algorithm
SRTM	Static Root of Trust Measurement
TBB	Trusted Building Block
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TSS	TPM2 Software Stack
TXT	Trusted Execution Technology
UEFI	Unified Extensible Firmware Interface
UI	User Interface
URL	Uniform Resource Locator
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNFM	Virtual Network Function Manager
WSN	Wireless Sensor Network

List of Figures

List of Tables

1 Introduction

The fast development and reduced cost of cloud computing has allowed many industries to deploy their systems in virtualized environments [14]. The telecommunications industry has shifted the deployment of their network functions from dedicated hardware to software modules that run on generic hardware. The version of cloud computing for telecommunications is called Network Function Virtualization (NFV), also known as the Telco Cloud [8], which is defined by the European Telecommunications Standards Institute (ETSI) [1] [15].

Network functions are distributed as one or more Virtual Machines (VMs), which run as virtual workload on servers. These functions, known as Virtual Network Functions (VNFs), provide critical functionality to the telecommunications systems.

The Trusted Computing Group (TCG) [2] has defined the specification for a microcontroller called the Trusted Platform Module (TPM) [49], which is often an embedded chip in the motherboard of a server. This chip can store confidential data, certificates, keys and cryptographic measurements of system components, including BIOS, bootloader and kernel. Additionally, it can generate keys and perform cryptographic functions. However, cryptographic functions will be done slowly, since the TPM is not a cryptographic accelerator. The latest library specification for TPM is version 2.0, which replaces the previous 1.2 specification [41]. TPMs provide a mechanism called quoting, which is used to report measurements of a platform to a third party, who can compare them to known good values to determine platform integrity. This process is called remote attestation.

There has been previous research on establishing machine trust at boot time and VNF trust at launch time by utilizing the TPM for integrity verifications and a third party Remote Attestation Server [43]. Trust is established by verifying that the underlying hardware platform is in a correct state. Ensuring correct platform state is crucial, given that VNFs control many critical network elements and functions, such as routers, firewalls, Evolved Packet Core (EPC) and Mobility Management Entity (MME). Therefore, we need to guarantee that the correct code is being executed in a safe environment.

There exist some remote attestation schemes in literature; however, the only open source remote attestation solution available for cloud platforms is Intel Open Cloud Integrity Technology (OpenCIT) [22]. OpenCIT leverages platforms with Intel processors, which include Intel Trusted Execution Technology (TXT) [20] extensions. Intel TXT allows establishing an environment in which software components can be measured as they are loaded and those measurements stored in the TPM. OpenCIT checks the integrity of a platform by utilizing a remote attestation server that requests the measurements stored in the TPM and compares them to known good values.

[1] https://www.etsi.org/

[2] https://trustedcomputinggroup.org/

However, this solution presents a few weaknesses, such as lack of flexibility on the definition of trust and the definition of trust being limited to a fixed point in time.

This work proposes a remote attestation solution, which can reason over platform trust in a finer-grained manner. Furthermore, it aims to provide a mechanism for determining the causes of trust failures and introducing mitigations for the affected device.

1.1 Problem statement

Integrity of a platform is usually measured at boot time, and those measurements do not change until the next reboot. If the underlying hardware is measured again during runtime, it will report the same measurements that were stored during boot time, even though the machine may have been compromised. Most of the servers running the virtual workload are not rebooted very often. Although there exists research that focuses on boot time integrity attestation[22, 42, 7], little effort has been devoted to include run-time integrity measurements in the attestation process.

Additionally, in current remote attestation schemes, the definition of trust is limited to a boolean value based on the state of a platform at a fixed point in time. TPMs contain a set of registers that can be extended with measurements. Existing solutions compare the contents of these registers to whitelisted values, if those values match, the platform is considered trusted, otherwise it is considered untrusted. Since many of the existing techniques use the outdated TPM 1.2 specification, these registers are the only information considered when determining platform trust. However, the newer TPM 2.0 specification includes a set of metadata that is reported along with system measurements. Including this metadata in the set of information used to determine platform trust would allow to reason over the system state in a finer-grained way.

Another limitation of existing attestation solutions is the lack of a notion of history for the platform measurements and trust status. This prevents reasoning over time about the trust state of a platform and determining its trustworthiness. For example, a platform that has been considered trusted every time it has been checked would be more trustworthy than a platform that has recently changed the status from untrusted to trusted. However, current attestation solutions would consider those two platforms equally trustworthy.

Similarly, trust status reasoning is limited to the information provided by the TPM. Other systems are not considered when determining platform trust. There are many events that may occur in the system that influence the trust status of a machine, such as software updates, machine reboots and changes in known good values. Current schemes do not take these events into consideration when reasoning over trust.

If a platform fails any check, it is considered untrusted and no workload will be

placed on it. However, there may be non-critical workload that would be suitable to run on platforms that only pass a subset of those checks. Also, a more flexible definition of trust can be used to allow machines with known faulty components to be a part of the trusted cloud. Little research has been directed at defining different degrees of trust in a cloud.

Finally, there are no forensics mechanisms in place to detect the reason behind attestation failures. One common thing to do when attestation fails is to prevent the machine from booting or isolating it from the rest of the network, which makes difficult diagnosing the reasons for the failure. Having a mechanism in place that would analyze failures and determine possible causes would allow administrators to have a deeper understanding of different error scenarios and introduce new actions to prevent or mitigate these situations.

1.2 Objectives and scope

This thesis has three main objectives.

1. Develop a fine-grained concept of trust in a cloud environment based on NFV.
2. Implement a policy and rule based mechanism to determine platform trust in a trusted cloud.
3. Propose a forensics system that allows identifying root causes for attestation failures.

In the scope of this thesis we focus on platforms with a TPM 2.0 on board. However, this work can be extended to include devices with older versions of TPM and devices or VMs without a TPM, e.g. by utilizing cryptographic hash measurements. However, the concept of a TPM and the need to establish a root of trust remain critical.

1.3 Contribution

The research carried out during this thesis resulted in two original publications.

The first publication [40] presents a summarized version of the work in this thesis, particularly the rule system described in Chapter 4. It discusses the use of remote attestation and RCA to determine system trustworthiness.

The second publication [39] combines different technologies to build a testbed for trusted telecommunications systems. The remote attestation solution designed and implemented in this work is used in the testbed for providing platform trust.

1.4 Thesis structure

The rest of this thesis is organized as follows. Chapter 2 introduces important background information for the understanding of this work, such as Network Function Virtualization and Trusted Computing, and reviews existing solutions. Chapter 3 presents the identified shortcomings in current remote attestation solutions. Chapter 4 describes the architecture and design of our proposed solution. Chapter 5 provides the implementation details of the system. Chapter 6 discusses the results of our work and outlines the directions this work can take in the future. Chapter 7 concludes the thesis work.

2 Trusted Computing Background

This chapter provides an introduction to Network Function Virtualization (NFV) and trusted computing. We discuss the proposed architectures for NFV and the Trusted Platform Module (TPM). Additionally, we discuss how the TPM can be leveraged to determine the integrity of a platform. Finally, we review previous work on remote attestation platforms.

2.1 Network Function Virtualization

Cloud computing has served as an enabler for the telecommunications industry to move their systems to virtualized environments. Traditionally, deploying network functions required specialized and proprietary hardware equipment. This equipment would need to be replaced often, due to fast advances in technology, which represents a high cost for companies that is not reflected in revenues [14]. Additionally, updating and managing deployed hardware represents a challenge, since many of these components are placed in locations that are hard to reach, such as cell towers.

Network Function Virtualization (NFV) [14], also known as the Telco Cloud [8], is the version of cloud computing for the telecommunications industry. It is defined by the European Telecommunications Standards Institute (ETSI) [15] as a reference architecture as shown in Figure 1. In NFV, the network functions are distributed as a set of Virtual Machines (VMs), which together form a Virtual Network Function (VNF). These VMs can be deployed as virtualized workload on traditional servers, which reduces the need for specialized hardware to run network functions. Furthermore, the cost for deploying, maintaining, upgrading and decommissioning network functions is reduced, since it can be done at the software level.

The reference architecture depicted in Figure 1 shows a set of components and how they interact with each other to build NFV. It consists of 5 main functional blocks: NFV infrastructure (NFVI), the virtualized network functions (VNFs), element management (EM), management and operations (MANO) and the operations and business support systems (OSS/BSS). The Network Function Virtualization Infrastructure (NFVI) contains all the hardware and software components necessary to deploy a VNF. It contains hardware resources, such as compute, storage and network. The NFVI is responsible for abstracting the hardware resources, so that the lifecycle of a VNF is independent of the hardware. It provides virtualized resources to the VNFs, and, from the perspective of VNF, it is a single entity providing these resources, instead of a pool of resources.

Virtualized Network Functions are software packages that implement a traditional network function, such as servers, firewalls and network elements. They are deployed on top of the NFVI layer, using the resources provided by it. The operations on a

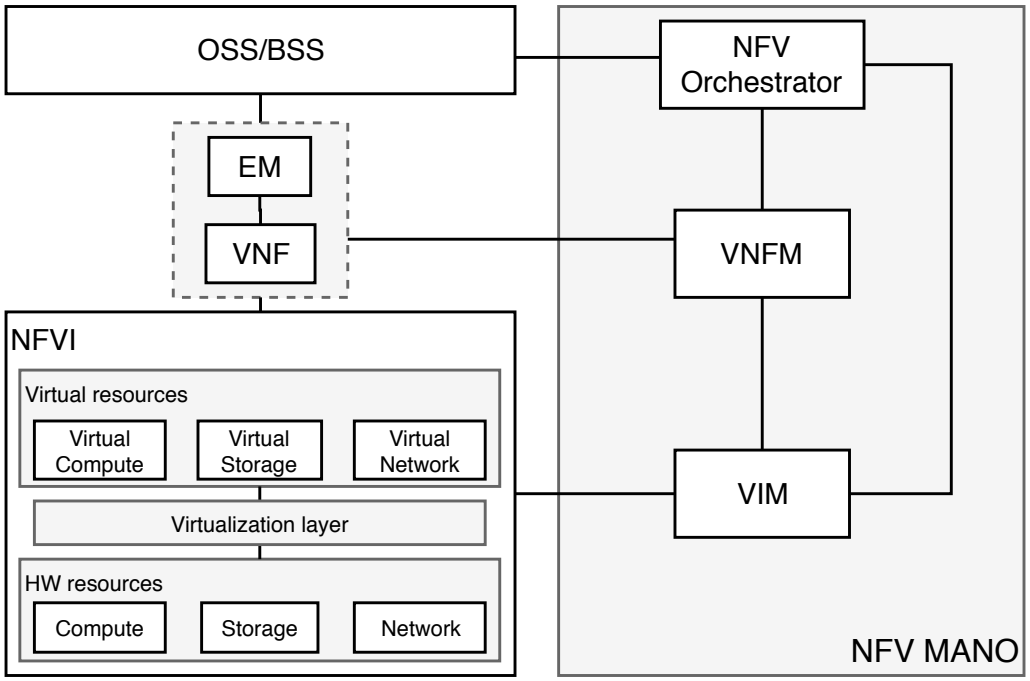

Figure 1: NFV Reference Architecture (Adapted from [15])

VNF are managed by the Element Management component.

The Management and Operations component comprises the Virtual Infrastructure Manager (VIM), Virtual Network Funtion Manager (VNFM) and NFV Orchestrator. The Virtual Infrastructure Manager handles all interaction between a VNF and the resources provided by the NFVI. The Virtual Network Function Manager is responsible for managing the lifecycle of a VNF, including instantiation, update and termination. The NFV Orchestrator is the component responsible for guaranteeing that there are enough resources available to provide a network service. To do so, it can interact with the VIM or directly with the NFVI.

Finally, the Operations and Business Support Systems (OSS/BSS) component refers to the OSS and BSS systems of a mobile network operator, which interact with MANO and VNFs to support the business operations.

Although there is a clear distinction between these components in the reference architecture, it is important to note that in practice the roles and functionality may overlap with one another or the functionality may be merged into a single component. For example, in OpenStack [3], both VIM and VNFM functionality are included in the Nova service component.

[3] https://www.openstack.org/

2.2 Trusted computing

Trusted computing denotes a set of technologies utilized to establish trust in a platform. It introduces trust anchors into the system and provides methods for verifying the integrity of a system. In this section, we discuss the architecture of the Trusted Platform Module (TPM), how it is used to establish a chain of trust for a platform and the attestation process utilized to verify the integrity of a system.

2.2.1 Trusted Platform Module

The Trusted Platform Module (TPM) is a microcontroller available in most server class hardware. It is a chip embedded in the motherboard of a server, which provides secure storage of keys, confidential data, certificates, cryptographic measurements of system components, as well as cryptographic functions and key generation. Figure 2 shows the main components in the architecture of the TPM version 2.0.

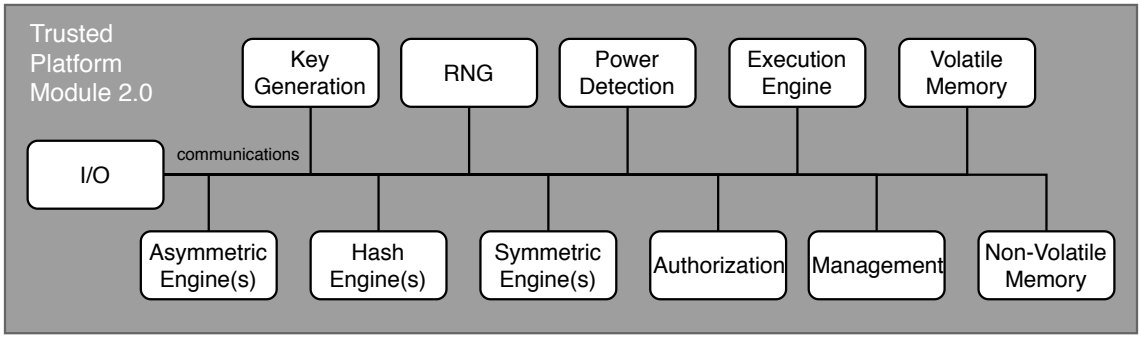

Figure 2: Trusted Platform Module architecture version 2.0 (Adapted from [49])

All interaction between the host system and the TPM is done through the I/O Buffer. The TPM includes three engines: asymmetric, symmetric and hash. They implement the algorithm(s) to be used by the TPM for asymmetric and symmetric cryptography, as well as hashing. The authorization component verifies, before running a command, that it has the correct authorization to execute. The power detection component gets notifications on power state changes and manages the TPM power states accordingly. The execution engine is the component in charge of executing the commands received. The random number generator (RNG) produces the randomness in a TPM by using entropy functions, state registers and mixing functions.

At manufacture time, each TPM is provided with a large, random value called a primary seed. This seed is stored on the TPM and cannot be retrieved. The key generation component can generate two kinds of keys: ordinary and primary keys. The former are generated using the RNG, whereas the latter are generated from a primary seed in the TPM. This component generates two key pairs: Endorsement Key (EK) and Attestation Key (AK), which give a TPM the notion of unique identity.

The EK is a storage key which is used as a primary parent key to generate new keys in the TPM. The AK is a restricted signing key, generated from the EK, which can only be used for signing data structures generated by a TPM, such as platform measurements.

Volatile memory stores transient data, such as platform configuration registers (PCRs), objects loaded to the TPM from external memory and session information. PCRs are registers used to store measurements of components of the system, including BIOS, kernel, hypervisor, and operating system. These measurements are cryptographic hashes of the components. On reset or restart, all PCRs are set to a default initial value. PCR values cannot be manually set, instead they must be *extended*. The extend operation works as follows:

$$\text{PCR}_{new} = \text{hash}(\text{PCR}_{old}||\text{new\_value})$$

where || denotes the concatenation operation.

Each TPM provides 24 registers (numbered 0-23) and can provide multiple banks of such registers depending on the algorithm used to extend the PCR. Our TPMs offer two PCR banks: SHA1 and SHA256.

Finally, Non-Volatile Random Access Memory (NVRAM) stores persistent data on the TPM. Part of the NVRAM can be allocated for use by the platform. One use for NVRAM is to store information and seal it against PCR values. When an NVRAM area is sealed against a set of PCRs, the contents can only be read when the PCRs are in the same state as when the area was sealed.

TPM Quotes The TPM provides a mechanism for obtaining measurements of the platform, called *quoting*. A quote can be requested for a set of PCRs. The TPM will generate a structure that contains a digest of the contents of the requested PCRs. This structure contains other metadata, such as a clock value, number of reboots and firmware version of the TPM. Table 1 includes the fields of the quote structure. The generated structure is signed with a restricted signing key (generally the AK). The TPM returns the structure and the signature to the requester, who now has the measurements for the system.

2.2.2 Measured boot

Measured or trusted boot is the booting process, in which every component in the boot sequence measures the next component in line before executing it.

In this process, a trust chain is built starting from the Core Root of Trust Measurement (CRTM). The CRTM is the first piece of code that is executed on platform boot.

Field	Description
attested	A hash over the values of the given PCRs
clock	Value of the TPM Clock at quote time
firmware	Firmware version of the TPM
magic	TCG defined magic value for a quote
qualifiedSigner	Name of the key used to sign the quote
resetCount	Number of power cycles/boots
restartCount	Number of suspend or hibernate events
safe	Denotes clock integrity
type	Header value for a quote

Table 1: TPM quote fields

It is referred to as a Trusted Building Block (TBB) by the TCG [52], and, since it is the root of the chain of trust, it should not change during the lifetime of the platform. The CRTM is usually stored in read-only memory in the BIOS boot block and is implicitly trusted.

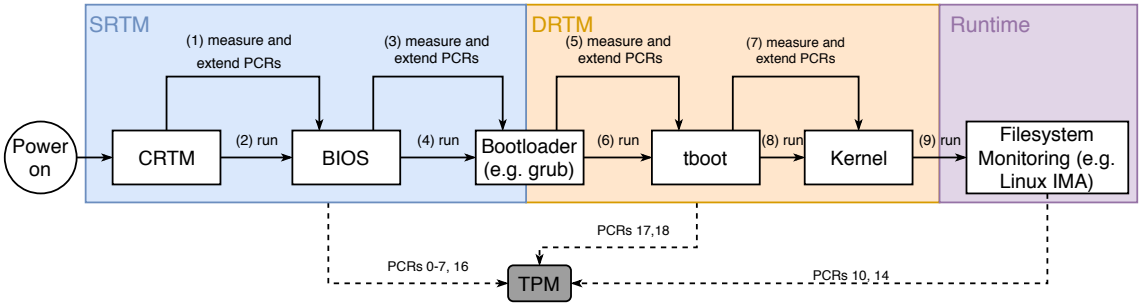

Figure 3: Chain of trust in measured boot

Figure 3 shows an example of the measured boot process, in which each component in the boot sequence measures the next before executing it. We can identify two different measurement stages: Static and Dynamic Root of Trust Measurement (SRTM and DRTM, respectively).

The STRM provides a chain of measurements for the components that are executed during boot before loading the operating system (OS), as well as their configurations. It provides an overview of the current state of the platform. The CRTM measures the BIOS and writes these measurements to PCRs 0-4. Then, the BIOS is executed, and it measures the bootloader. These measurements are stored in PCRs 5-7. Debug measurements may be stored in PCR 16.

The DRTM provides a chain of measurements after the OS has booted. It contains measurements related to the kernel that will be loaded. In order to generate DRTM measurements, tboot [23] should be placed in the first slot of the bootloader. Tboot is a tool that can be placed between the bootloader and the operating system, to provide measurements of the OS kernel to be executed. It will execute an instruction from Intel TXT [20] to start a measured launch environment (MLE). In this mode, the processor is limited to a single core and the memory is locked so that only

authenticated code can run. Intel offers a set of Authenticated Code Modules (ACMs), which initialize the platform to a well-defined state and are the root of the DRTM trust chain. After the ACM runs, the control is given back to tboot, which measures the kernel to be launched by the bootloader and stores these measurements in PCRs 17 and 18.

PCR	**Usage**
0	CRTM, SRTM, BIOS, Host Platform Extensions, Embedded Option ROMs and PI Drivers
1	Host Platform Configuration (BIOS settings)
2	UEFI driver and application code
3	UEFI driver and application configuration and data
4	UEFI boot manager code (usually the master boot record) and boot attempts
5	Boot manager code configuration and data and GPT/Partition table
6	Host Platform Manufacturer Specific
7	Secure Boot Policy
10	Linux IMA
16	SRTM Debug
17	ACM, MLE, tboot policy and kernel measurements (DRTM)
18	Public key used to sign the ACM, tboot policy and control values (DRTM)

Table 2: PCR usage during measured boot

Table 2 shows a summary of the usage of PCRs in a measured boot. Since PCRs can only be extended, these measurements are taken by extending the previous PCR. For example, the value of PCR 1 will be: $extend(PCR0, new\_measurements)$. The extend operation guarantees a chain of trust, since each PCR measurement depends on the previous one in the chain. Therefore, if a malicious component is present in the boot sequence, its measurements will be included in one of the PCRs. The chain of trust will break at that point, since the measurements will be different than the ones expected, and the component will not be able to forge the measurements for the subsequent PCRs.

Measured boot should not be confused with UEFI Secure Boot [56]. In secure boot each component in the boot sequence is signed by a trusted signer. Then, each component verifies that the signature of the next component in the chain is valid before executing it. If the signature of a component is invalid, it is not executed. Note that no measurements are taken of the components, instead only signature checks are performed. Therefore, secure boot cannot provide proof of the state of a platform, it can only guarantee that the components executed were signed by a trusted party.

In measured boot, it is possible to determine the state of the platform by the measurements taken during boot. Measured and secure boot have different approaches to unknown components in the boot chain. Measured boot will continue the boot process even in the presence of an unknown component, whereas secure boot will prevent their execution.

2.2.3 Remote attestation

Remote attestation is the process in which a challenger can check the integrity of a platform with the help of an attestation server. By using measured boot, all platforms with a TPM will be able to provide measurements that indicate the state of the platform. In a cloud environment, it is desirable to check if a platform is trusted or not before launching virtual workload on it. A simple definition of trust states that a machine is trusted when its measurements match a set of good expected values.

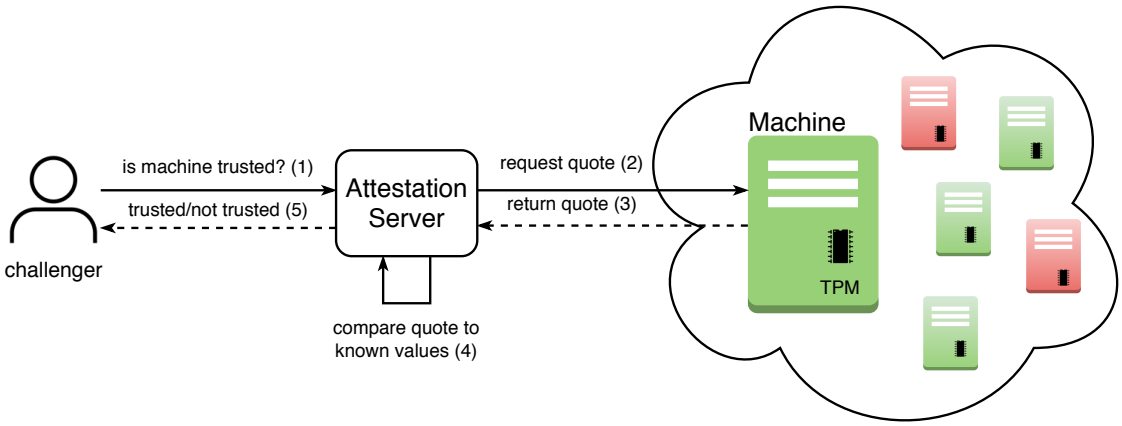

Figure 4: Remote attestation process

Figure 4 shows the remote attestation process. In this scenario, a challenger requests the attestation server to provide a report of the trust status of a machine. The attestation server is a trusted third party, who stores the set of correct known values for each machine in the cloud. It can interact with the machine and request a quote of its measurements. The machine provides the quote, and the attestation server proceeds to compare the quote to the known values for that machine. This comparison determines whether the machine is trusted or not. Finally, the attestation server responds to the challenger with the trust status of the machine. In Section 2.4, we review some existing implementations of remote attestation services in practice and literature.

2.3 Linux Integrity Measurement Architecture

The Linux Integrity Measurement Architecture (IMA) [46] is an integrity subsystem in the Linux kernel. It introduces hooks within the Linux kernel to measure the integrity of a set of files in the filesystem. Files are measured before they are read or executed, and these measurements are stored into a log that can be consulted by administrators. Additionally, if a TPM is available, an aggregate integrity value over

the list of measurements is stored in PCR 10. Any PCR can be used to store this integrity value; however, 10 is the *de facto* standard.

IMA uses policies to determine which files are measured in the system. The default policy aims to measure the Trusted Computing Base (TCB) of the system, which is defined as the set of components that are critical to the security of a system [45]. The IMA TCB policy measures all files in the system that are considered sensitive: executables, libraries mapped to memory and files opened for read by root.

Linux IMA can be used to provide run-time integrity attestation. IMA can detect changes in a file at run-time, since it re-measures each file before reading or executing it.

IMA policies can be extended. The kernel documentation includes instructions on how to write custom policies [59]. In order to define finer grained policies, IMA can leverage the use of file metadata maintained by Linux Security Modules (LSMs). One useful LSM is Security Enhanced Linux (SELinux) [48], which keeps an object type field in the metadata of each file in the system. SELinux allows the creation of new custom types. Therefore, we can create a custom type to tag a set of interesting files for runtime attestation and define an IMA policy over files with this type tag. IMA policies allow users to indicate custom PCRs to store the integrity value over the measurements taken by the policy.

```
1  ...
2  measure func=frame=single, FILE_CHECK obj_type=measure_t pcr=14
3  ...
```

Figure 5: Example line in custom IMA policy

Figure 5 shows an example of an entry in an IMA policy that indicates that all files tagged with type `measure_t` by SELinux should be measured, and the integrity value over these measurements should be stored in PCR 14.

2.4 Existing solutions

There is a wide variety of work related to trusted computing and remote attestation in literature. In this section, we discuss existing work related to attestation.

Jacquin *et al.* [27] introduce a remote attestation solution for verifying the integrity of Software Defined Networking (SDN) switches and virtual machines, which execute critical network functions. Their approach uses a verifier, similar to an attestation server, which communicates with each monitored device and verifies the state of the platform. This work focuses on extending the capabilities of the TPM to VMs by using Linux IMA to measure the configurations that are monitored. However, this

approach uses a limited definition of trust at the platform level, since they evaluate trust in a pointwise manner with the results stored in the TPM from measured boot. One drawback of this approach is that the TPM version used was a TPM 1.2, which is now deprecated.

Similarly, Xu *et al.* [57] present a mechanism to establish trust between two NFV platforms by verifying the platform configurations. They use Linux IMA to measure the platform configurations and store an integrity value over the measurements on the TPM, as described in Section 2.3. When a platform wants to collaborate with another platform to provide a network service, the first platform can request the measurements for the configuration in the second platform and determine whether it is trusted or not. This work uses an outdated TPM 1.2 simulator. Moreover, they only focus on inter-platform trust by verifying NFV configurations. The authors do not discuss in detail the trust establishment process on a platform based on measured boot.

Extending remote attestation to other architectures has been a topic for extensive research. With the increasing popularity of the Internet of Things (IoT) and embedded devices, research has focused on how to extend trusted computing to these kind of devices, which often use ARM architectures or are resource constrained [18, 47, 2, 12, 3, 29, 9]. However, this research does not include TPMs, instead they utilize technologies such as ARM TrustZone security extensions [4]. Other work has focused on using TrustZone to emulate TPM-like functions [31]. Finally, there is also work in which TPMs are used for attestation on Wireless Sensor Networks (WSNs) [19] and mobile devices [37, 13].

Although it is possible to find multiple research papers related to remote attestation, much of this work focuses on the outdated TPM 1.2 version or schemes without a TPM. The only open source remote attestation solution available for cloud platforms is the Intel Open Cloud Integrity Technology (OpenCIT) [22], which is the successor of the OpenAttestation project [24]. Existing work on establishing cloud trust [42, 7, 36] utilizes OpenCIT and builds on top of it.

Intel CIT provides an attestation service for establishing trust in a cloud environment. It provides an attestation server, which communicates with the monitored platforms to determine their trust status, this process is similar to the one described in Section 2.2.3. Additionally, when starting a VM, it adds the possibility to query the attestation server for the status of a platform, so that VMs are only started on platforms with an appropriate trust attestation.

OpenCIT integrates with OpenStack cloud deployments by extending certain OpenStack components. These extensions allow the user interface (UI) to display the attestation results for each monitored machine and add the trust filter into openstack for VM placement. Unfortunately, OpenCIT suffers of some of the problems identified in Chapter 1 and discussed in detail in Chapter 3. These problems include a limited definition of trust and the lack of consideration of previous attestation results and

other events in the system, when determining current platform status. Moreover, it seems that OpenCIT is not being actively developed, since the last commit on the latest release of the project is over one year old (June 2017) [4].

2.5 Summary

In this chapter, we discussed the concept of Network Function Virtualization and the reference architecture for it. Additionally, we introduced trusted computing concepts, such as the trusted platform module, how it is used for measured boot and the process of remote attestation. Finally, we discussed existing work related to remote attestation and introduced an existing open source solution for remote attestation in the cloud.

[4] https://github.com/opencit/opencit/commits/v3.2.1

3 Problem Statement

This chapter analyzes identified limitations in current remote attestation schemes. The problems discussed in this chapter served as motivation for the design of the remote attestation solution presented in this thesis.

3.1 Run-time integrity

In the Telco Cloud, VNFs are run as virtualized workload that uses resources provided by the NFVI. The NFVI component of NFV comprises a set of hardware platforms, which are commonly generic server class hardware. One goal of establishing trust in NFV is to guarantee the integrity of the platform running the VNFs.

Boot time integrity has been achieved by using measured boot in previous work [21, 42, 7, 36] and using the methods described in Chapter 2. This kind of integrity measurements guarantee the trust chain up until the kernel loading stage. Using only boot time measurements can represent a challenge, as mentioned by the author in [43], since the same measurements taken during boot are stored in the TPM until the next boot. Therefore, the measurements taken some time after boot may become outdated.

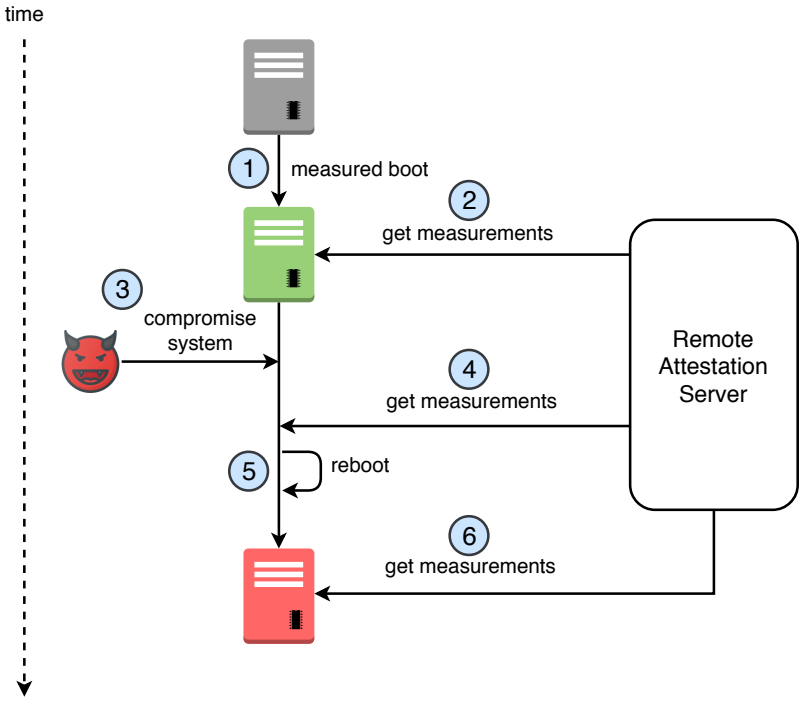

Figure 6: Boot time measurements and remote attestation

Figure 6 shows an example situation in which boot time measurements are not enough

to reliably establish platform trust. In this example, steps 1 and 2 show the measured boot and remote attestation processes. In step 3, an attacker manages to compromise the system, e.g. by installing a rootkit. We assume the server is not rebooted in the time between the attack and the next attestation. The next time the remote attestation server requests the measurements from the system, the measurements will appear to match the known values, since they are the measurements taken last time the machine was booted. Therefore, this malicious software will not be detected at least until the next time the machine is rebooted (steps 5 and 6). This example makes it clear that we need to include run-time measurements, in order to have more reliability in the remote attestation results.

Moreover, there are other parts of the system that should be measured during run-time to guarantee they have not been tampered with, e.g. hypervisor, critical software components running as bare metal processes and system configuration files. Recent research [44] discusses possible attack vectors in NFV by compromising the hypervisor running the virtual machines.

There exists research that covers run-time integrity measurements [57, 21, 27, 58], but it focuses on measurements of the virtual workload on the cloud platform. Guaranteeing the integrity of VMs and VNFs continues to be a popular topic of research. However, in this thesis we focus on establishing trust in the underlying NFVI platform, which is not as widely discussed in literature and remains an open challenge.

3.2 Trustworthiness history

In a remote attestation scheme, the trust in a platform is evaluated by measuring its components and comparing these measurements to known values. When a challenger requests the trust status of a device, the attestation server obtains the latest measurements of the platform and evaluates these against known values to determine the attestation results, which are then returned to the requesting party. However, the requesting party only has access to the latest attestation results, since existing schemes do not store a history of the measurements and attestation results for each machine they monitor.

Although the challenger is usually interested in the latest state of the platform, current schemes have the disadvantage that they are not able to provide a trust history for a device, which would determine its trustworthiness. We consider the trustworthiness of a machine to be determined by the amount of time it has remained in a trusted state. By not keeping a history of the attestation results, current remote attestation solutions are not able to assess the trustworthiness of a machine.

Figure 7 shows an example in which the trust status of two devices (machine 1 and 2) is evaluated over time. We can see that machine 1 is always considered trusted,

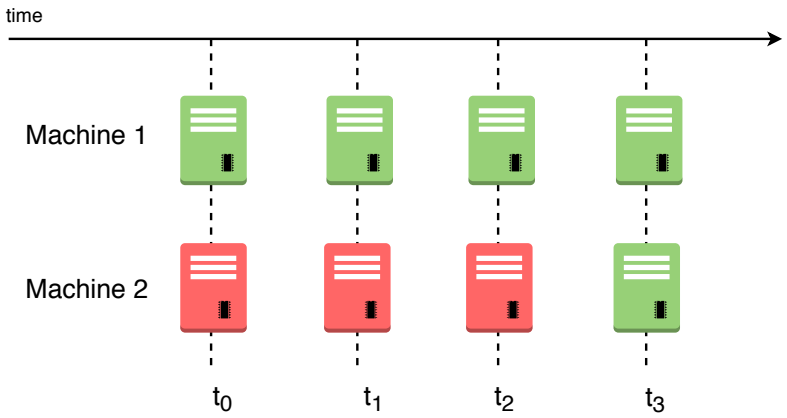

Figure 7: Trustworthiness of two machines

whereas machine 2 is only considered trusted for the latest results (at time t_3). If the attestation server has no notion of history for the measurements taken and trust status, machine 2 will be considered just as trustworthy as machine 1, although machine 1 has been trusted for a longer period of time. We wonder if the notion of history, regarding measurements taken and trust status of a machine, would improve future decision making of the attestation server. For example, by combining the trust history of an element with events in the system (see Section 3.4).

3.3 Limited definition of trust

The trust status of a machine is determined by comparing the measurements of its components (stored in the PCRs of the TPM) against known good values from a reference machine. In existing work, these measurements are the only information considered when determining if a platform is trusted or not. However, the TPM 2.0 quote structure that is returned when requesting machine measurements provides extra metadata, which can be used to aid in deciding on a device's trust status. This metadata is part of the quote structure as a set of fields with relevant information about the platform (e.g. firmware version of the TPM, clock value and amount of reboots). These fields were described in Chapter 2, Section 2.2.1.

A large amount of existing research utilizes the outdated version 1.2 of the TPM, which does not include these extra fields in the quote structure. Therefore, the trust decisions can only be based on PCR values. The remote attestation schemes that utilize TPM 2.0 still do not consider the extra fields in the quote when determining trust.

These fields in the quote can prove useful to detect unusual behaviour in the devices monitored by the attestation server. Take as an example a machine which reports correct measurements every time it is quoted. However, every time we obtain a new quote, the reboot count field of the quote reports an increase of 10 in the counter.

This would indicate that the machine has been rebooted 10 times between quotes, which would be considered as unexpected behaviour for servers in a datacenter and would require further investigation.

Another useful field in the quote structure is the firmware version of the TPM. A recent vulnerability was discovered for a certain version of the TPMs firmware, which makes RSA keys generated by the TPM insecure [1]. A patch for the firmware of the TPM was released to fix this vulnerability. The firmware version field of a TPM would allow us to verify that the TPM in our system runs a patched firmware without this vulnerability. In a system which does not consider the extra data, a TPM with a vulnerable firmware would go unnoticed.

Current remote attestation schemes show a limitation by not utilizing the extra information provided by the TPM, which can be used to create a more comprehensive definition of trust.

3.4 Considering other systems

In many cloud scenarios, there are other deployed systems, which could provide useful information to the remote attestation process, such as a patch management system (PMS) or an intrusion detection system (IDS). Current solutions are not designed to integrate output of these systems into the attestation process. Components external to attestation provide context for the events happening in the system. Furthermore, they can help drive the attestation process.

In the cloud, the machines in the NFVI layer need to be patched periodically. Usually, there is a system in charge of managing the upgrade process, which includes releasing patches and triggering updates. If this component can provide input to the attestation server, we can use it to trigger attestation evaluations.

Figure 8 illustrates how an external system, such as a patch management, can provide useful information for the attestation process. If we assume the external component can provide input to the attestation server, when a patch is released for a particular machine, the attestation server can be notified and take measurements of the machine before the update. After the update, the attestation server can quote the machine again to obtain the latest measurements. This approach would allow the attestation server to verify that both a patch was applied and that it was the correct patch.

Unfortunately, current solutions perform remote attestation in a periodic manner, since they are not designed to receive input from external systems which could drive the attestation process.

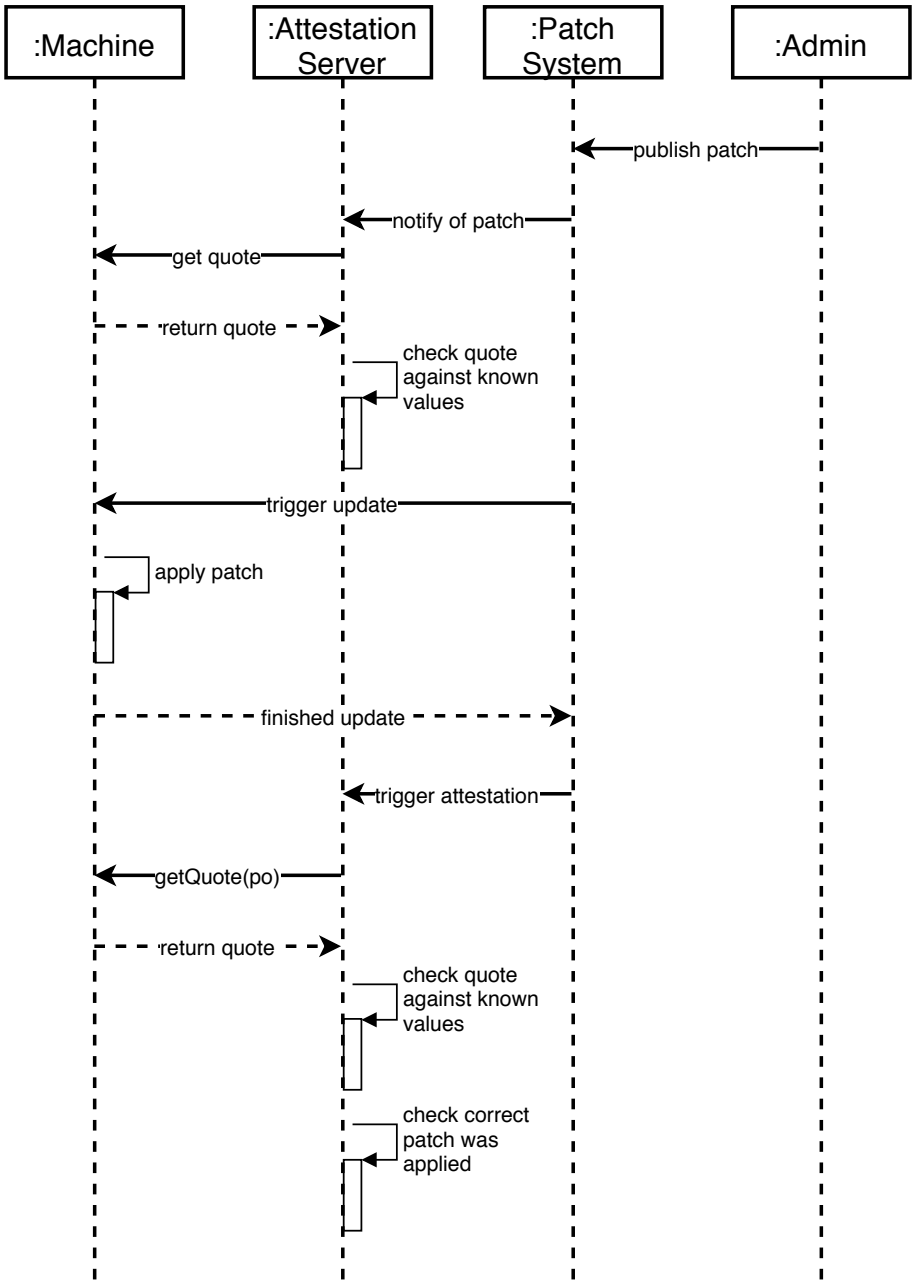

Figure 8: Using PMS to trigger remote attestation

3.5 Platform resilience

In existing solutions, when attestation fails, the device is considered untrusted, and it is not used again until the measurements match the known values again. Moreover, there are no different degrees of trust for a machine. A machine can only be trusted or not trusted, depending on whether its measurements match the known values or not.

For example, if a server in the NFVI layer that is used to provide virtual resources to the VNF layer fails its trust check, no virtual workload will be deployed on this machine. This may be the correct behaviour for critical workload, since we do not want to run our VNFs in potentially unsafe platforms. However, there may be situations in which having a weaker definition of trust would allow us to still use the server to deploy non-critical workload, thus increasing the resilience of the system, which is considered a key requirement in NFV [16].

Current work considers trust at a single level. However, we consider that remote attestation can benefit from defining a trust hierarchy which determines different degrees of trust in a system. Given the previous example, upon a trust failure, the attestation server could re-evaluate the trust of a machine according to less strict criteria defined by the trust hierarchy, until the machine reaches a trusted state. Then, depending on the degree of trust of the machine and the requirements of the workload, it may be possible to still utilize the resources provided by the platform.

3.6 Analysis of attestation failures

When a machine is considered untrusted as a result of attestation, it is not enough to isolate the machine until the measurements return to correct values. Ideally, we want to set mitigations in place for the machine to recover from the failure. Additionally, we want to investigate the causes for the failures, in order to prevent them in the future, if possible.

In current solutions it is only possible to check what PCR measurements do not match the correct values known by the attestation server. Current solutions do not provide a mechanism that would allow an administrator to analyze attestation failures for a machine. However, there is no mechanism which would allow an administrator to analyze and determine the underlying cause of the failure.

We consider that existing work can be extended by defining a procedure or system, which would allow a system administrator to investigate and understand attestation failures. Furthermore, the attestation failures can be linked to the actions that need to be taken, in order to recover the trust in the system.

3.7 Summary

In this chapter, we identified and discussed in detail a set of shortcomings encountered in existing work. The drawbacks described here were considered when designing the architecture of the solution this work proposes.

4 Architecture and Design

This chapter describes an architectural overview of the remote attestation solution introduced in this work. We have developed a remote attestation server, which can monitor elements in a trusted cloud. Furthermore, this attestation server is extended with a rule system, which allows to define a finer-grained definition of trust. We have designed a set of agents that sit on the monitored machines. These agents are used to provide information about the machine, e.g. measurements and identity, as well as events that have happened, e.g. reboot events, to the attestation server. We explain each component of the architecture as shown in Figure 9.

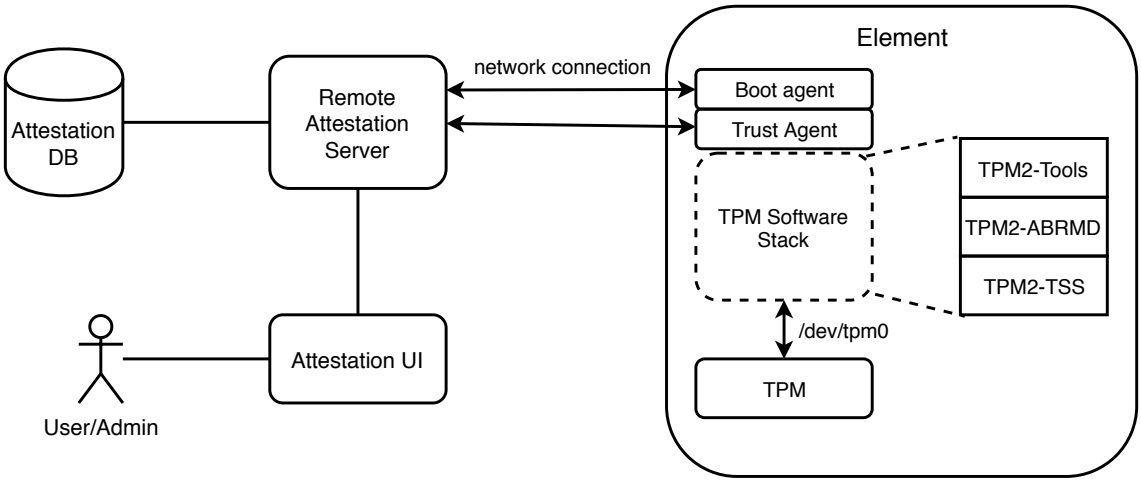

Figure 9: Remote attestation architecture (for an element with a TPM)

4.1 TPM tools stack

In order to allow our elements to interact with the physical TPM, we need to set up and configure a software stack for the TPM. This stack consists of three components: a library for interacting with the TPM, a resource manager to handle multiple accesses to the TPM and a set of tools to get the TPM functionality. These are explained in detail in the next sections.

4.1.1 TPM2 software stack (tpm2-tss)

The TPM2 Software Stack (TSS) [53] is a set of components which handle all the low-level interaction with the TPM. This software stack implements a set of APIs for interacting with the TPM. It provides four layers, which implement each an application programming interface (API). At the upper layer it provides a System API (SAPI) and an Enhanced System API (ESAPI), which implement the functionality of all commands that can be sent to a TPM. The next layer

is the Marshalling/Unmarshalling API, which provides functions for constructing byte streams to send to the TPM, as well as decomposing the response streams. Finally, there is the TPM Command Transmission Interface (TCTI), which provides a standard interface to send and receive TPM commands. The TSS implementation is available on GitHub [5].

4.1.2 TPM2 access broker & resource management daemon (tpm2-abrmd)

This is a system-wide daemon that provides two functionalities: resource management and access brokering [51]. The resource manager functionality acts as a virtual memory manager for the TPM. Since the memory on the TPM is limited, the resource manager is in charge of swapping objects in and out of memory as needed, so they are available for use on the TSS level. The access broker functionality handles the synchronization between different processes that use the TPM. It guarantees that no process is interrupted when performing an operation on the TPM. The implementation of the daemon is available on GitHub [6].

4.1.3 TPM2 tools (tpm2-tools)

These are a set of command line tools used to interact with the TPM. These tools can either communicate directly with the TPM device or use the resource manager described in the previous section. The functionality provided includes quoting, listing the contents of the PCRs, signing, managing keys, creating policies for sealing and interacting with NVRAM. The implementation of the tools is available on GitHub [7]. The GitHub page also includes a detailed description of the commands implemented.

4.2 Machine agents

Machines in our system are provided with a set of agents that handle the communication with the attestation server. We have defined two agents, which are described in the next sections.

4.2.1 Trust agent

The trust agent is a process that runs on every element, which serves as a communication link between the TPM and the attestation server. It uses the TPM tool stack

[5] https://github.com/tpm2-software/tpm2-tss
[6] https://github.com/tpm2-software/tpm2-abrmd
[7] https://github.com/tpm2-software/tpm2-tools

to obtain information from the TPM and report it back to a third party (usually the attestation server). The information the trust agent can report to the attestation server includes element quotes, TPM capabilities, NVRAM areas defined on the TPM, system information about the element, EK and AK of the TPM.

4.2.2 Boot agent

The boot agent is a process that runs on every element, which serves as a source of information about the reboot events on an element. It provides information about the system state to the attestation server. The main responsibility of the agent is to report boot events to the attestation server. This agent will alert the attestation server whenever the element is shutdown or started up.

4.3 Attestation database

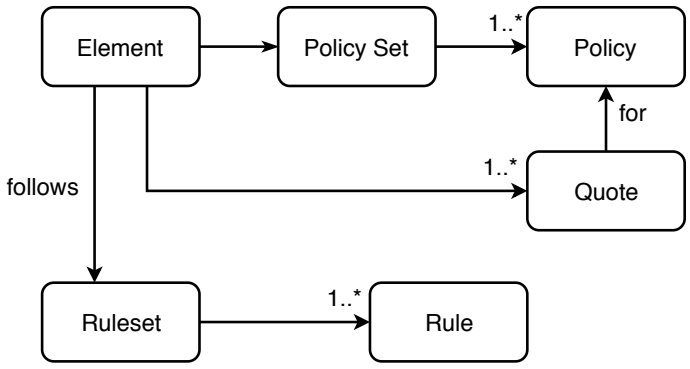

Figure 10: Relationship between different attestation data structures

The attestation database stores information about the elements in the system, the existing policies, their measurements and the different rules used to determine trust of an element. It also stores information about events that happen in the system. Figure 10 shows how the different data structures are related to each other. Elements may have one policy set associated to them, which is composed by one or more policies. Similarly, they may have one or more quotes taken for them. The quotes are always taken for one specific element and one policy. Finally, the elements follow one ruleset, which is composed of one or more rules. The rules and rulesets will be explained in more detail in Section 4.6

In the following sections, we will explain in detail the definitions of elements, quotes, policy sets and policies.

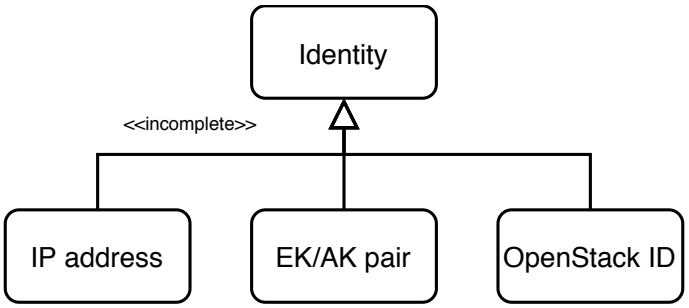

Figure 11: Types of identities

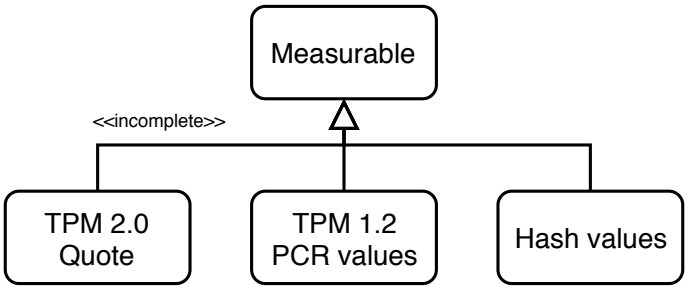

Figure 12: Types of measurements

4.3.1 Elements

An element is any device that is *attestable*. We consider an element attestable when it can be uniquely identified and measured. The attestable elements in our cloud include servers, laptops and IoT devices. Figure 11 shows some of the different notions of identity our elements have. Note that this is not a comprehensive list. Some of these identities are more permanent than others, e.g. IP addresses and OpenStack IDs may change easily, whereas the endorsement and attestation key pairs cannot be changed unless the TPM is replaced on the element.

Similarly, Figure 12 shows some of the different types of measurements we can take of an element. Most of our elements are measurable using TPM 2.0, but other ways of measuring elements include TPM 1.2 and hash values. Again, this list is incomplete and can be extended to include other forms of measurements.

Table 3 shows the information fields stored in the database for an element.

4.3.2 Quotes

The process of obtaining measurements from a device with a TPM is called *quoting*. A quote is a data structure generated by the TPM upon request for measurements for a set of given PCRs. This structure contains a hash over the values of the requested PCRs, as well as some interesting metadata, which includes the number of reboots,

Field	Description
_id	Unique ID given to the element on the database
ek	Public part of the Endorsement Key of the TPM
ak	Public part of the Attestation Key of the TPM
ip	IP address
kinds	List of human readable type names/tags for this element, e.g. `Element::TpmMachine`
last_trust_decision_event	ID of the event that contains the last trust decision made for the element
name	A human readable name for the element
openstack_id	Unique identifier given to the element by OpenStack
policies	List of policies this element is associated with
ruleset_id	ID of the ruleset that must be evaluated to determine the trust status of the element
status	Indicates whether an element is trusted or not
timestamp	Indicates the time that the element was last updated
uname	System information obtained by executing `uname -a`

Table 3: TPM element fields

number of suspensions, a clock value and the firmware version of the TPM. Table 4 shows the fields included in the quote data structure.

Additionally, the quote structure is signed by either the Attestation Key of the TPM or a given suitable signing key.

Field	Description
_id	Unique ID given to the quote on the database
element_id	ID of the element for which the quote is
kind	A human readable type name, e.g. `Quote::TPM2.0`
quote	A dictionary containing the quote structure from the TPM (see Table 1)
pcrs	A dictionary containing the PCR values at the time of the quote
timestamp	Indicates the time that the quote was added to the database

Table 4: Quote fields

4.3.3 Policies

Policies are a mapping between a possible measurement that can be taken from an element (e.g. a set of PCRs) and the correct expected value for that measurement. Policies are built based on a reference machine by quoting it for a set of PCRs and storing the PCRs and quote results. The attestation database stores these measurements and the expected values. Therefore, next time an element is quoted, the value can be compared to the expected one on the database.

Table 5 shows the information fields stored in the database for a policy.

Field	Description
_id	Unique ID given to the policy on the database
pcrs	Measurement that can be taken from a TPM element, in this case a list of PCRs
expected_value	Correct value for the measurement described in the PCRs field
kind	A human readable type name, e.g. `Policy::TPM2.0`
name	A human readable name for the policy
timestamp	Indicates the time that the policy was last updated

Table 5: TPM policy fields

4.3.4 Policy sets

A policy set is a collection of policies for a particular element. Each policy contains a measurement to be taken for a specific component of an element, e.g. CRTM, SRTM or DRTM. A policy set combines all the policies that belong to a single element. Policy sets are used by the attestation server to determine what measurements to request from the trust agent running on the element. Table 6 shows the information fields stored in the database for a policy set.

Field	Description
_id	Unique ID given to the policy on the database
policy_ids	IDs of the policies in this policy set
kind	A human readable type name, e.g. `Policy::PolicySet`
name	A human readable name for the policy set
timestamp	Indicates the time that the policy set was last updated

Table 6: TPM policy set fields

4.4 Attestation server

The attestation server is the component of the system responsible for obtaining measurements from elements and determining their trust status based on predefined policies. The attestation server interacts with the attestation database to store and retrieve information necessary for managing the trust in the cloud. It handles all operations over the items stored in the database and described in this chapter: elements, policies, quotes, policy sets, rules, rulesets and events.

The attestation server knows how to communicate with NFVI elements to request measurements. It can receive event notifications from the elements, such as updates or reboots. Additionally, it can communicate with NFV MANO elements to provide information about the trust status of NFVI elements. It can be queried by third parties and asked to evaluate the trust status of a platform. The information obtained from the attestation server can be used by other systems. For example, the trust status of a platform can be used as an extra criterion by OpenStack to decide on which machine to deploy virtual machines on.

The rule system described in Section 4.6 is a component within the attestation server in charge of determining the trust status of an element. Finally, the attestation server provides the information that is displayed by the attestation UI.

4.5 Attestation UI

The attestation UI is the component responsible for providing a human friendly overview of the cloud status to an user or administrator. It is used as a dashboard for the cloud administrator to quickly determine the overall trust status of the cloud. Additionally, it contains information about the system known by the attestation server, such as elements, policies, policy sets and quotes. Furthermore, it includes detailed reports on how the trust of an element was determined by running a ruleset against an element and its policy set. It shows what rules were evaluated, and, if the trust checks fail, it allows the user to determine why.

4.6 Rule system

We introduce a rule system to reason over the definition of trust for elements in our cloud infrastructure. This rule system contains individual rules that evaluate to a boolean value and rulesets that combine these rules, in order to create different definitions of trust.

4.6.1 Rules

The rules reason over an element and a policy. Each rule has an *apply* function that runs the rule and returns a Boolean value, which indicates if the rule is followed or not. The result of running a rule is added to the attestation database as an event.

Static rules A static rule reasons over a quote for a policy and an element at a specific point in time. The quote over which the static rules reason is usually the latest quote taken for that element and policy. The element must follow the policy for which it is quoted, and the quote must be for that specific policy.

TPM quotes include a set of values that can be used to determine if an element is trusted at a given point in time. Static rules reason over these values for a single quote and compare them to correct expected values.

Given an element e, a quote q and a policy p for q, and assuming that e follows p,

we define the logic of a static rule called "Correct attested value" in Equation 1.

$$\text{correctAttestedValue}(q, p) = \begin{cases} \text{True}, & q.attested = p.expectedValue \\ \text{False}, & \text{otherwise} \end{cases} \tag{1}$$

This rule compares the 'attested' field of the quote to the expected value indicated in the policy stored in the attestation database. The 'attested' field contains a digest of the contents of the given PCRs. The policy for this quote stores the PCRs and the expected 'attested' value for that PCR combination. The rule will be satisfied when the attested value matches the expected value, and it will fail when it does not match.

Appendix A contains a detailed description of all the static rules defined in our rule system.

Temporal rules Our attestation database stores quotes taken for an element over time. This allows us to include temporal reasoning into our definition of trust, since we can monitor the changes on specific parts of quotes in a specific time frame. Furthermore, we can reason over changes in different parts of the system (e.g. patch management, virtual infrastructure management and networking) in the time interval between two quotes.

We define rules that reason over two quotes for a policy and an element. These two quotes are usually the two latest quotes; however, they can also be any pair of quotes given that one happens at some point in time before the other one. Since all the quotes in the attestation database are timestamped, it is trivial to select the two latest quotes. There are some known behaviours along time for different fields of a quote. We can use the temporal rules to verify that the values are changing, or not changing, as expected.

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of two temporal rules over the 'resetCount' field of a quote in equations 2 and 3.

$$\text{resetCountNotChanged}(q, q') = \begin{cases} \text{True}, & q.resetCount = q'.resetCount \\ \text{False}, & \text{otherwise} \end{cases} \tag{2}$$

$$\text{resetCountIncreased}(q, q') = \begin{cases} \text{True}, & q.resetCount < q'.resetCount \\ \text{False}, & \text{otherwise} \end{cases} \tag{3}$$

The 'resetCount' field of a quote is a counter that increases every time an element is rebooted [49]. It is reset to zero when the TPM is cleared. During the normal

lifecycle of an element, the reset count should either stay the same or increase. If the reset count decreases, it may be an indicator that the TPM was reset to the factory defaults.

Additionally, we define rules that reason over the events that happened between a pair of quotes. Since the attestation database stores all boot and update events for an element, we can define temporal rules that check if an element has changed between quotes, e.g. updated or rebooted. Similarly, we store any update events to the policy of the element, which allows us to detect policy changes between quotes.

Given an element e and pair of quotes for e, q and q', assuming q was taken before q', we define the logic of the temporal rule that checks if an element has been updated in Equation 4.

$$\text{elementUpdated}(e, q, q') = \begin{cases} \text{True}, & (\exists\ u \in \text{getEvents}(q.timestamp, q'.timestamp) \mid \\ & \quad u.type = \text{`element update'} \wedge u.element = e) \\ \text{False}, & \text{otherwise} \end{cases}$$

$$\text{where:}$$

$$\text{getEvents}(t, t') = \{e \mid t \leq e.timestamp \leq t' \wedge e \text{ is an event in the attestation DB}\} \quad (4)$$

Every time an element is updated, we store an 'update' event in the attestation database. Similar to the previous rule, this rule reasons over 'update' events for an element that may have happened between two quotes. Combined with other rules, it allows us to detect situations in which changes in a quote are due to a software update on the element. This rule will be satisfied when the element has been updated between quotes, and it will fail otherwise.

Similarly, given a set of reboot events B and a pair of quotes for e, q and q', assuming q was taken before q', in Equation 5, we define the logic of a temporal rule that checks if the amount of reboots reported by an element matches the amount of reboot events stored in the database for the time interval between q and q'.

$$\text{resetCountMatchesReboots}(q, q', B) = \begin{cases} \text{True} & (q'.resetCount - q.resetCount) = |B| \\ \text{False} & \text{otherwise} \end{cases} \quad (5)$$

This rule will be satisfied when the amount of reboots reported by an element matches the reboot events between quotes and fail otherwise.

Appendix B contains a detailed description of all the temporal rules defined in our rule system.

Compound rules Finally, we define a set of compound rules. A compound rule reasons over a group of rules, which can be static or temporal. These kind of rules allows us to build more complex reasoning over the simple rules we have defined so far.

On their own, static and temporal rules do not have enough information to determine trust status. There may be situations in which one of the previous rules will fail if considered individually. However, when we combine the output of different rules, we can build a more detailed context to make a decision.

There are two basic types of compound rules: AND and OR rules. Given a set of rules R, we define the logic of an AND rule in Equation 6 and the logic of an OR rule in Equation 7. An AND rule will evaluate all its children rules and return $True$ if all the children rules were satisfied. On the other hand, an OR rule will evaluate all its children rules and return $True$ if at least one of the children rules was satisfied. Note that we can build trees with these rules, since compound rules can be both parent and children rules.

$$\text{AndRule}(R) \quad = \quad \begin{cases} \text{True} & (\forall\ rule \in R \mid rule.result = True) \\ \text{False} & \text{otherwise} \end{cases} \tag{6}$$

$$\text{OrRule}(R) \quad = \quad \begin{cases} \text{True} & (\exists\ rule \in R \mid rule.result = True) \\ \text{False} & \text{otherwise} \end{cases} \tag{7}$$

We introduce an example of a compound rule for checking the amount of reboots for an element. The TPM keeps a reset counter that tracks the amount of times the element has been rebooted. Our system stores boot events every time an element in the trusted cloud reboots. This rule checks that the amount of reboots reported by the TPM and the attestation database are the same. It uses the result of the rules: "resetCountNotChanged" (described in Equation 2), "resetCountIncreased" (Equation 3) and "resetCountMatchesReboots" (Equation 5). Given a pair of quotes q and q' (assuming q was taken before q'), and a set of reboot events B during the time interval between q and q', we define the logic of the rule in Figure 13.

This compound rule is an OR rule, since there are two possible situations to evaluate: when the machine has rebooted and when it has not. In the case when the machine has not rebooted, we check that the TPM reset count has not changed. On the other hand, if the machine has rebooted, we check that the TPM reset count has increased and that the reset count matches the amount of reboot events in the attestation database. This rule will be satisfied when the machine has not been rebooted, or when there have been reboots and the amount of reboots matches the reboot events

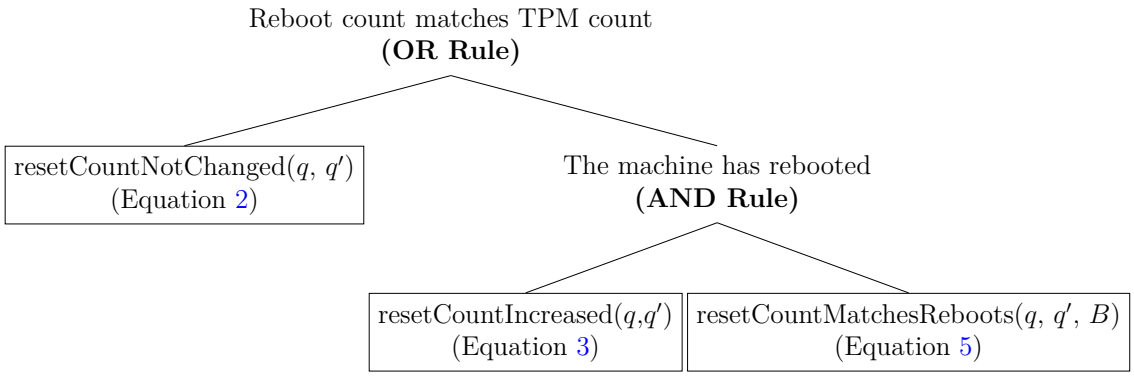

Figure 13: Example of a compound rule (Reboot count matches TPM count)

in the database. It will fail if the TPM reset counter has decreased or if the amount of reboot events does not match the counter.

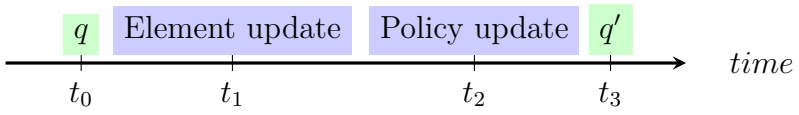

Figure 14: Element was updated and policy changed between quotes

Compound rules can be used to define how trust should be evaluated in complex scenarios. Figure 14 shows the case when the element gets a software update, which affects one of its policies, and the policy is updated to reflect this change. Both updates happen between quotes q and q'. In this situation, the attested value changes; however, since the policy was updated, the latest quote will still satisfy the policy. Figure 15 shows the logic for the rule. Note that the equations for some of the children rules are defined in Appendix B.

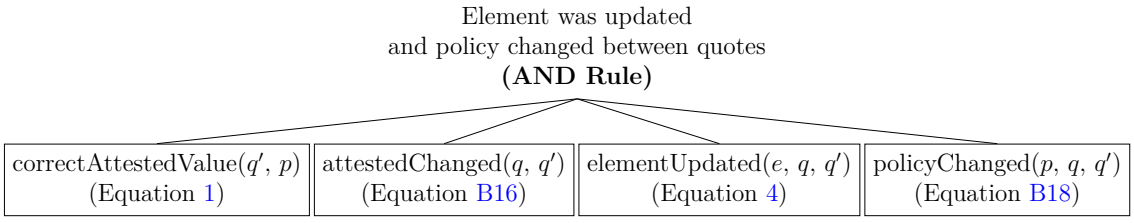

Figure 15: Element was updated and policy changed between quotes (AND Rule)

Therefore, if an element has been updated between quotes, but the policy has been changed to reflect this update, we can still consider the element trusted.

4.6.2 Rulesets

The rules described previously can be combined in different rulesets. These rulesets define trust. For an element to be trusted against a ruleset, all the rules in the ruleset must evaluate to $True$.

Elements are associated with a ruleset and a policy set. For an element to be considered trusted, the ruleset must be evaluated against all the policies in the policy set the element is associated to, and the rules must evaluate to $True$. Rulesets have an apply functionality, which runs the rules in the ruleset against an element and a policy. They also have a decision functionality, which applies the ruleset against an element and all the policies in the policy set. Finally, the ruleset generates a trust decision event that is stored on the attestation database and updates the trust status of the element.

Rulesets can be constructed in such a way that their definition of trust can vary in strictness. Furthermore, a partial order can be derived over the different rulesets defined. In this work, we use the rules defined previously to construct four rulesets (in ascending strictness order):

Minimal This ruleset contains the minimum amount of rules that need to be run to consider an element trusted at a single point in time. It checks that the quote satisfies the policy, the signature is valid, the firmware is one of the known ones and the magic and value types are correct. The rules included are defined in the following equations in appendix A: A1, A2, A7, A5 and A6.

Minimal + clock increasing This ruleset extends the "Minimal" ruleset by adding a temporal check over the value of the clock. It checks the same conditions as the minimal ruleset. Additionally, it checks that the clock is increasing between quotes. The rules included are defined in the following equations in appendices A and B: A1, A2, A7, A5, A6 and B5.

Minimal + clock integrity This ruleset extends the "Minimal" ruleset by adding temporal and integrity checks over the value of the clock between quotes. It checks the same conditions as the minimal ruleset. Additionally, it checks that the clock is increasing between quotes. If the clock is decreasing between quotes it checks whether the clock value is reliable or not, based on the safe value reported by the TPM. The rules included are defined in the following equations in appendices A, B and C: A1, A2, A7, A5, A6 and C2.

Extended This ruleset extends the"Minimal + clock integrity" ruleset by adding additional temporal checks to the trust evaluation. Additional checks include detailed reboot checks, to determine if the element has been rebooted or suspended, and if so, checks that the TPM counters reflect the possible changes in state (e.g. upon reboot, the reset count increases). In addition, this ruleset checks that the firmware version of the TPM has not changed between quotes. The rules included are defined in the following equations in appendices A, B and C: A1, A2, A7, A5, A6, C2, C6 and B3.

Extended + reboot comparison This ruleset expands the "Extended" ruleset by adding a more detailed check on the reboot count of the element. It includes rules that cross check the amount of reboots reported by the TPM with the amount of reboots stored in the attestation database. The rules included are defined in the following equations in appendices A, B and C: A1, A2, A7, A5, A6, C2, C6, B10 and B3.

Complete This ruleset expands the previous one by adding checks over more fields of the quote and detailed checks to the attested value, which consider update events happening in the system. It checks the behaviour of those fields which are supposed to change between quotes (quote and signature) and those which do not (magic, type and qualified signer). Additionally, it checks the events in the attestation database to explain changes in the attested value. The rules included are defined in the following equations in appendices A, B and C: A1, A2, A7, A5, A6, C2, C6, B10, B3, B17, B7, B1, B4, B2 and C17.

4.7 Attestation forensics and root cause analysis

We have now defined a set of rules that can be evaluated at any time against an element to determine its trust status. Usually, an element is bound to fail the trust checks at some point during its lifecycle. Using Root Cause Analyis (RCA), we can map these failures to Causal Factor Trees (CFT) or Ishikawa diagrams [26], which contain possible causes for the failure. Furthermore, we can map these trees to mitigation procedures, which is a step towards automating the recovery from trust failures. These mitigation procedures can be adapted to address the identified failure situations and the probability of the event [10].

Our Causal Factor Trees are diagrams inspired by fault tree analysis [54, 25] and why-because analysis [32]. These analysis techniques identify factors surrounding the fault or failure and show the causality relationships between such events. Fault tree analysis combines the events by using logical gates, whereas in why-because analysis an edge from one node to another directly indicates causality. In our CFTs, the root node is the failure detected, and all the branches of the tree are possible causes for the failure. Each child node answers to the question "why?". If a node has more

than one child it is because there are many possible causes for that failure. The leaf nodes contain the root causes for the failure in the topmost node.

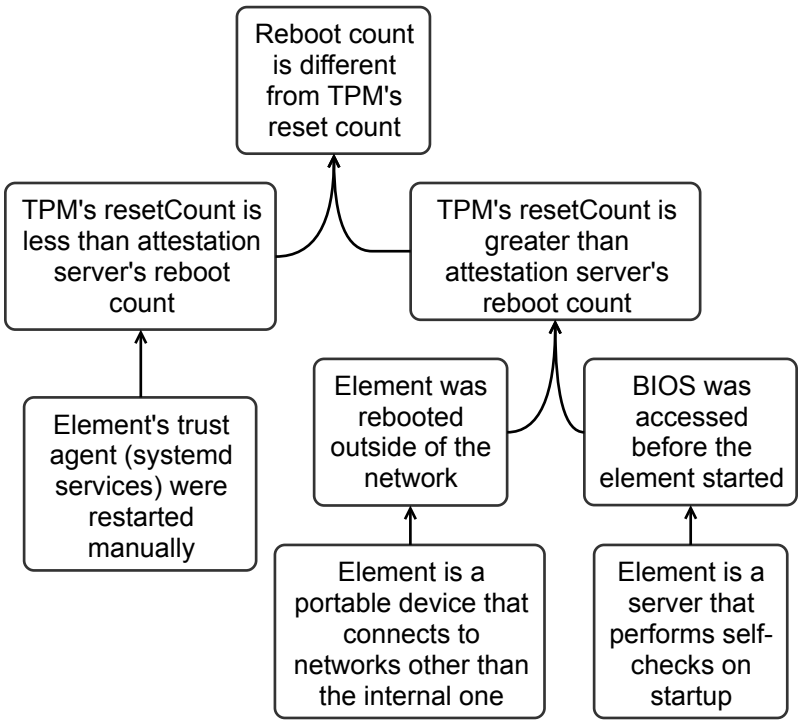

Figure 16: Causal factor tree for reboot trust failure

Figure 16 shows a CFT constructed for the scenario in which an element fails its trust check, due to an inconsistency in the amount of reboots reported by the TPM and the ones stored on the attestation database.

The amount of times a machine has been rebooted is counted in two ways in our system. One of them is the reset count reported by the TPM on a quote event, which is considered to be accurate. The second way is by using the boot agent installed on the element. Occasionally, we can detect a mismatch between these two counters. However, there may be a valid reason for this.

There are two possible failure situations: the TPM can either report a lower or a higher amount of reboots than the ones stored in the attestation database. In the first case, a cause for this failure may be a manual restart of the agents of the element (boot or trust agent). Since these are systemd services, they can be manually restarted, which would report a reboot event to the attestation database that did not really happen. In the second case, we have identified two possibilities for the count reported by the TPM to be higher, one detected when using portable devices and the other one detected when using server class hardware.

Portable devices, such as laptops, are usually moved in and out of the internal network. When these elements are outside of the internal network, they cannot report their reboots to the attestation database. Therefore, it is possible that the

elements reboot outside of the network, in which case the TPM would record these events but the attestation database would not.

On the other hand, we don't expect servers to move outside of the internal network. However, this kind of hardware often includes self-checks on startup. These self-check procedures may include multiple reboots before the boot loader is executed. Once more, in this situation, the TPM will have an account of these reboots that the attestation database fails to record.

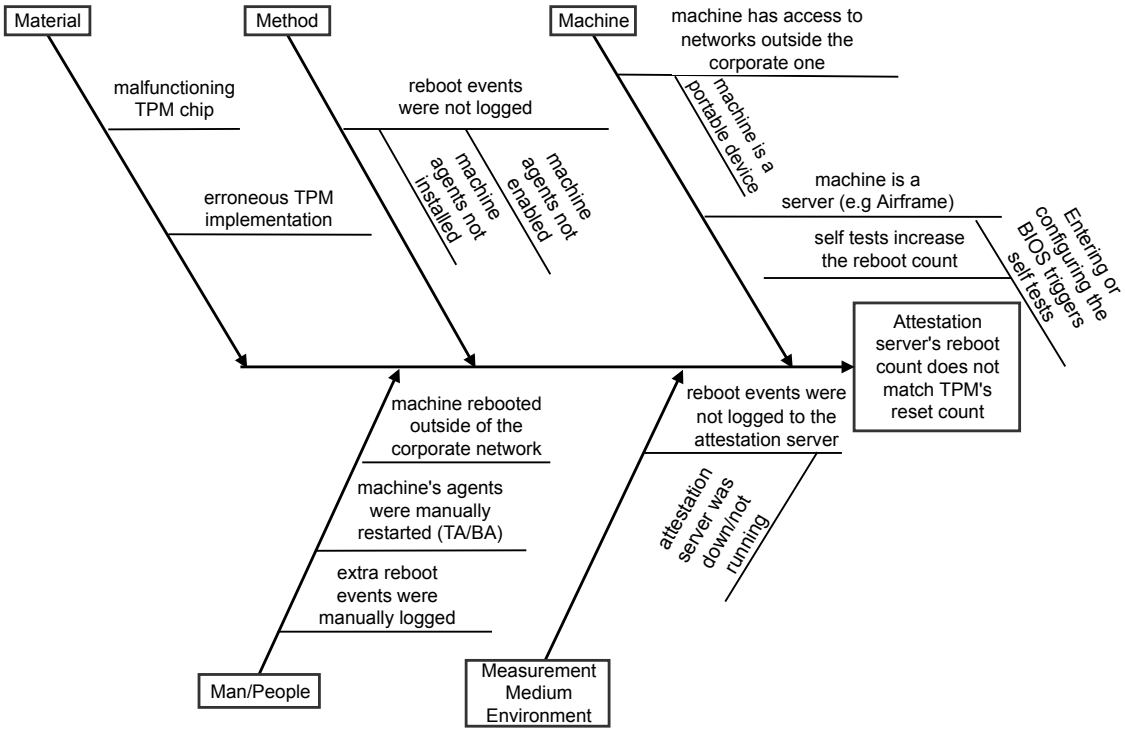

Figure 17: Fishbone diagram for reboot trust failure

In an Ishikawa or Fishbone diagram the "why?" question is answered in a different way. Each diagram has five main bones that represent a different aspect or part of the system. The main bones contain smaller bones, which describe possible causes for the failure that relate to the part of the system the main bone describes.

Figure 17 shows the same trust failure scenario described using an Ishikawa or fishbone diagram. This kind of RCA diagram allows us to differentiate between different parts of the system when analyzing the causes for a failure. It allows to individually analyze the areas that may be responsible for the failures, such as the hardware or software of element in question, the attestation server, the network, the actions of users and the methods used by the attestation procedures set in place.

The diagram in Figure 17 has five main bones: Material, Method, Machine, Man/People and Measurement/Medium/Environment.

The 'Material' bone represents the hardware of the element in question. Under

this branch, we describe all possible causes that can be attributed to an element's hardware. In this particular case, we identified two possible hardware causes for the reboot count mismatch: a malfunctioning TPM chip or an erroneous TPM implementation.

The 'Method' bone includes all the processes followed in the system. Under this branch, we describe all possible causes that can be attributed to the way things are done. In our case, we identified one possible cause: reboot events not being logged. Then, we have two possible sub-causes for reboot events not being logged in the system: the machine agents (trust and boot agents) are either not installed or not enabled.

The 'Machine' bone comprises both the behavior of the element and the software in the element. Under this branch, we identified two causes, which have their own sub causes. The first cause is the element having access to outside networks, due to being a portable device. The second cause is the machine being a server, which means entering the BIOS when rebooting triggers self testing, and self testing increases the reboot count.

The 'Man/People' bone includes all the causes related to actions of users of the system. Under this branch, we identified three causes: the element being rebooted outside of the corporate network, the machine agents being manually restarted and the manual logging of additional reboot events.

Finally, the 'Measurement/Medium/Environment' bone includes all causes related to other subsystems the element interacts with. Under this branch, we identified one possible cause: the reboot events not being logged to the attestation server, due to it being unavailable.

Many of the causes identified in the bones of the Ishikawa diagram are similar. The separation of these into different branches allow us to identify the aspect of the system in which the failure may be corrected. This kind of diagram can be helpful when designing mitigation procedures to the trust failures, since it would help identify in what area these should be placed.

This failure is only one of the many trust failures we have identified in the proof of concept of this work. Appendices D and E include a list of these Causal Factor Trees and Ishikawa diagrams.

4.7.1 Extending rules and rulesets

Root Cause Analysis with Causal Factor Trees and Ishikawa diagrams allow us to identify situations in which trust or trustworthiness of an element can be safely recovered. We can consider the situation in which the TPM of an element reports a higher count of reboots than the ones in the attestation database, from Figures 16

and 17. We noticed that this behavior was normal in server class hardware. Therefore, we can identify the servers in our cloud that show this kind of behavior and create a new rule.

Given a pair of quotes q and q' (assuming q was taken before q'), and a set of reboot events B during the time interval between q and q', we define the logic of the rule in Equation 8.

$$\text{resetCountHigherThanReboots}(q, q', B) = \begin{cases} \text{True} & (q'.resetCount - q.resetCount) \geq |B| \\ \text{False} & \text{otherwise} \end{cases} \tag{8}$$

We can add the rule described in Equation 8 to the ruleset of the element or class of elements. Next time the element is attested, a higher reboot count from the TPM will not make this element not trusted, since this will be considered normal behavior.

RCA allows us to detect failures and set the correct mitigations in place. Furthermore, combined with our rule system, we can modify and introduce new rules to create a finer-grained definition of trust. This definition of trust can change from one element to another, which allows us to deal with atypical behavior that does not affect trustworthiness of an element.

4.8 Event system

Our attestation server takes into consideration events that happen in the system when determining the trust status of an element. These events and relevant information about them are stored in the attestation database, to allow them to be checked when making a trust decision. Events can be for a specific element, from the attestation server or even from other system components. Table 7 shows the common fields for all events stored on the attestation server.

Field	**Description**
event_type	A human readable name that describes the event type
result	Boolean value that indicates if the event was successful or failed
error_msg	Optional message describing an error. It is included when the result is False
_id	Unique ID for the event on the database
timestamp	Time when the event was stored in the database

Table 7: Common event fields

In the next sections, we describe the structure of the main types of events used by the rule system.

4.8.1 Boot events

Each element has a software component, called the boot agent, which communicates with the attestation server. Every time that element reboots, the boot agent will send an event to be stored in the attestation database with details about the reboot. The event_type field for these events is "Boot event". Additionally, boot events contain an extra field called element_state which can be "START" or "STOP" depending on whether the element has been shut down or powered on. One STOP and one START boot event pair are considered one reboot for an element. Finally, they include an element_id field that indicates the ID of the element being rebooted.

4.8.2 Element software updates

Each element has a software component, which communicates with the attestation server whenever an element is updated. When an update occurs, the element will inform the attestation server and an update event will be stored on the attestation database. The event_type field for these events is "Command run". They include three extra information fields, which are described in Table 8.

Field	Description
event_id	ID of the element being updated
command	Command used to update the element, e.g. `apt-get update`
output	Output of running the command

Table 8: Additional fields for element software update events

4.8.3 Attestation database updates

Similarly, when an element, policy or policy set stored on the attestation database is modified or updated, the attestation server stores an update event. These events are particularly useful to keep track of how these elements change over time, since it may be relevant for creating rules and rulesets. They include three extra information fields as shown in Table 9

4.8.4 Rule run events

Every time a rule is run, an event is generated and stored on the attestation database. These contain information about the rule, such as for what element, policy and quotes the rule is evaluated and the result of evaluating the rule.

Element update	
Field	**Description**
element_id	ID of the element being updated in the database
old_element	Copy of the element stored in the database before the change
new_element	Copy of the element stored in the database after the change
new_parameters	Parameters to be updated
Policy update	
Field	**Description**
policy_id	ID of the policy being updated in the database
old_policy	Copy of the policy stored in the database before the change
new_policy	Copy of the policy stored in the database after the change
new_parameters	Parameters to be updated
Policy set update	
Field	**Description**
policy_set_id	ID of the policy set being updated in the database
old_policy_set	Copy of the policy set stored in the database before the change
new_policy_set	Copy of the policy set stored in the database after the change
new_parameters	Parameters to be updated

Table 9: Additional fields for attestation database update events

4.8.5 Ruleset run events

A ruleset contains a collection of rules that are applied to an event and a policy. As mentioned before, when an individual rule is evaluated an event is generated. A ruleset keeps a list of the IDs of such events. After all the rules in a ruleset are run, an event is stored in the database that includes the result of running the ruleset against a particular element and policy, as well as a list of event IDs for each of the rules included in the ruleset.

4.8.6 Trust decision events

To evaluate the trust status for one element, we need to run the ruleset against all the policies in the policy set of the element. A trust decision event is generated after running a ruleset against an element and its policy set. It contains the final trust status for the element, the ID of the element, the ID of the policy set, the ruleset used to determine the trust status and the list of IDs for each ruleset run event generated after running a ruleset against a policy. This trust decision event contains all the information used to determine the trust status of an element.

4.9 Attestation server in the ETSI NFV architecture

Our system introduces the attestation server as a new component to the ETSI NFV reference architecture. Figure 18 shows the modified architecture, which includes the attestation server. We consider the attestation server as a MANO element, which provides interfaces to communicate with the VIM, VNFM, VNFs and NFVI. NFVI includes the physical servers on which the VNFs are run. VNFs can be deployed as one or more VMs on the servers provided by the NFVI. For simplicity reasons, in this work, we assume that a VNF is always deployed as a single VM. Since, in practice, many of the VNFs are considered a single VM, we consider this to be a reasonable assumption.

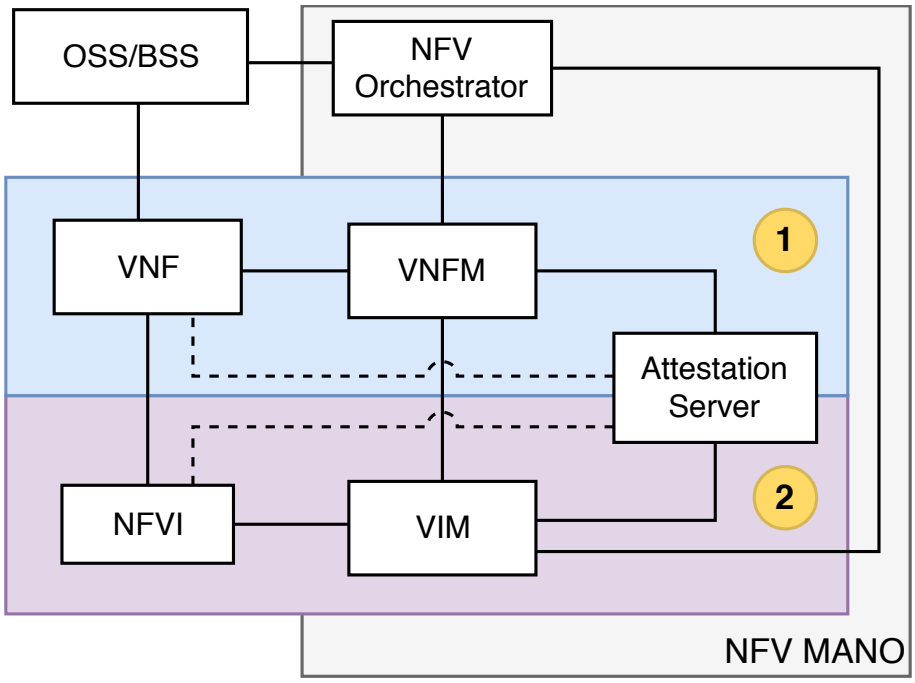

Figure 18: Remote attestation server in the NFV Architecture

We identify two main scenarios in which the attestation server is used (denoted 1 and 2 in Figure 18). In Scenario 1, the attestation server communicates with the VNFM and VNFs to provide information about the trustworthiness of the underlying NFVI layer or the other VNFs. In Scenario 2, the attestation server interacts with the VIM and NFVI elements. The NFVI elements can interact with the attestation server to report measurements, reboot events or software updates, by using the machine agents. Similarly, the attestation server can communicate with the VIM to obtain information about the NFVI elements, such as confirmation about the reboot events or IP addresses. Scenario 2 is the main focus this thesis; although, the attestation server was built to easily support and explore Scenario 1.

4.10 Summary

In this chapter, we introduced the architecture of our system and discussed its components in detail. Additionally, we described the design of our rule system, which aims to introduce flexibility to the definition of trust for an element. Furthermore, we discussed the integration of root cause analysis to our system, to obtain attestation forensics. Finally, we discussed the modifications to the ETSI NFV architecture to include the attestation server.

5 Implementation

This chapter discusses the implementation of the architecture described in Chapter 4. We include implementation details of the attestation server, tools, UI and machine agents. Finally, we discuss particular use cases in the system.

5.1 Trusted infrastructure

Our cloud consists of a set of elements that are monitored by the attestation server. Table 10 shows a summary of these elements and their specifications. Our trusted cloud includes two Nokia Airframe servers, three Intel Next Unit of Computing (NUC)[8] mini PCs and two laptops. The servers are used as compute nodes in our local trusted cloud deployed on OpenStack. The VNFs in the cloud would be deployed on these servers. The NUCs are used for experiments in the testbed as IoT devices. Finally, the laptops are for regular use, e.g. work and Internet browsing.

Element	OS	Kernel	TPM Firmware
Compute (Nokia Airframe)	Ubuntu 17.10	4.10	1407374883832066
Compute2 (Nokia Airframe)	Ubuntu 17.10	4.10	1407374883832066
NUC1 (Intel NUC)	Ubuntu 16.04	4.4	1407641172325888
NUC2 (Intel NUC)	Fedora 27	4.16.7	1407641172325888
NUC3 (Intel NUC)	Fedora 27	4.16.7	1407641172325888
Lenovo Thinkpad X1 v5	Ubuntu 18.04	4.15	1970586830692608
Lenovo Thinkpad X1 v6	Ubuntu 18.04	4.15	19984776037404676

Table 10: Trusted cloud elements

These elements are configured with different releases of Ubuntu and Fedora, as well as different kernel versions, since we wanted to test our solution on different environments. Additionally, the TPMs in our elements have slightly different versions, which causes occasional changes in behaviour from one platform to another, due to implementation differences.

We have enabled Intel TXT and the TPM in the BIOS for all elements in our cloud, to obtain SRTM measurements. Additionally, we have installed tboot and modified our bootloader (GRUB [9]), to load tboot first, to obtain DRTM measurements. With this configuration, all our elements are capable of performing measured boot as described in Section 2.2.2.

Also, we have set up each device with the TPM 2.0 tools stack 4.1, which includes the libraries to interact with the TPM, the resource manager daemon and the command

[8] https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html
[9] https://www.gnu.org/software/grub/index.html

line tools. Figure 19 shows the result of executing the `tpm2_pcrlist` tool, which prints the contents of the PCR banks, after a successful measured boot. We can see that PCR 0 contains CRTM measurements, PCRs 0-7 and 16 contain SRTM measurements, PCRs 17 and 18 contain DRTM measurements and PCR 10 contains measurements for the TCB as defined by Linux IMA.

```
$ tpm2_pcrlist
sha1 :
  0  : ae62d40b59561f52419a6dbb1fe3f589e7c67856
  1  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  2  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  3  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  4  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  5  : 870a58e79a43e9dcdd273b6efa9975d882ce7420
  6  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  7  : 4037336fa7bc0eabe3778fcfff5fcd0ee6adcde3
  8  : 0000000000000000000000000000000000000000
  9  : 0000000000000000000000000000000000000000
  10 : 6d90af5442e1d4e24609f15cd5e2a8e96773d015
  11 : 0000000000000000000000000000000000000000
  12 : 0000000000000000000000000000000000000000
  13 : 0000000000000000000000000000000000000000
  14 : 0000000000000000000000000000000000000000
  15 : 0000000000000000000000000000000000000000
  16 : f0e5a9f7d5a33a281fff32e565cb407df5b191c1
  17 : 5ffa409f3ddbb8b339d9e7e9fbcec4a38d6dc817
  18 : 9b8547051d0e68839490e82f99a17d78e3e94fed
  19 : 0000000000000000000000000000000000000000
  20 : 0000000000000000000000000000000000000000
  21 : ffffffffffffffffffffffffffffffffffffffff
  22 : ffffffffffffffffffffffffffffffffffffffff
  23 : 0000000000000000000000000000000000000000
sha256 :
  0  : d8a76f44656e5b7ed75ddc6c19071d8594e99edb67c54c0f5f562a8bdaa26bbf
  1  : 3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
  2  : 3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
  3  : 3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
  4  : 3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
  5  : c38c9900b691c6f2d8b5c4959e3548a14b7ba61713a194560c545c62c4b0b4a5
  6  : 3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
  7  : b5710bf57d25623e4019027da116821fa99f5c81e9e38b87671cc574f9281439
  8  : 0000000000000000000000000000000000000000000000000000000000000000
  9  : 0000000000000000000000000000000000000000000000000000000000000000
  10 : 0000000000000000000000000000000000000000000000000000000000000000
  11 : 0000000000000000000000000000000000000000000000000000000000000000
  12 : 0000000000000000000000000000000000000000000000000000000000000000
  13 : 0000000000000000000000000000000000000000000000000000000000000000
  14 : 0000000000000000000000000000000000000000000000000000000000000000
  15 : 0000000000000000000000000000000000000000000000000000000000000000
  16 : 4ca16202817eb69aba12b94c1410ba17f0d136879377bf9aeb62a117597870e2
  17 : 13271ef5faf9c806e07d33ca81837b7bb8bf2e112e72b7ee3d8e8fe0b8bb784e
  18 : 3d403e3eb24aa3a73a766f9782ab728bc9a94c5c11dab7734729aff2fdba2336
  19 : 0000000000000000000000000000000000000000000000000000000000000000
  20 : 0000000000000000000000000000000000000000000000000000000000000000
  21 : ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
  22 : ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
  23 : 0000000000000000000000000000000000000000000000000000000000000000
```

Figure 19: PCR contents after measured boot

5.2 Machine agents

Each device is provisioned with a trust and a boot agent, which are necessary for interacting with the attestation server. Both agents are implemented using Python 2.7 [10] and deployed on the element as systemd [11] services.

5.2.1 Trust agent

The trust agent is the component in charge of providing information about the device to the attestation server. It can report three sets of information to the attestation server: identity, quote/measurements and capabilities. It is implemented as a Flask [12] server, which provides a REST API (Representational State Transfer Application Programming Interface) used by the attestation server to request information. It is started as a systemd service after booting the device. Table 11 shows the endpoints of the trust agent API.

URL	Operation	Description	Result
/identity/(key_handle)	GET	Obtain identity information of the element	Dictionary containing EK, AK and extra data
/capabilities/(key_handle)	GET	Obtain TPM capabilities	Dictionary containing the TPM capabilities
/quote/(pcr_string)/(key_handle)	GET	Obtain a quote for a set of PCRs	TPM quote and PCRs
/keys/(key_handle)	GET	Obtain a public key stored in the TPM	Requested public key
/get_ev/(pcr_string)	GET	Obtain the expected attested value for a set of PCRs	Attested value

Table 11: Trust agent REST API endpoints

Each endpoint takes a `key_handle`, which is the handle of a key in the TPM. This key will be used to sign the response from the trust agent. Additionally, they take a

[10] https://www.python.org/
[11] https://www.freedesktop.org/wiki/Software/systemd/
[12] http://flask.pocoo.org/

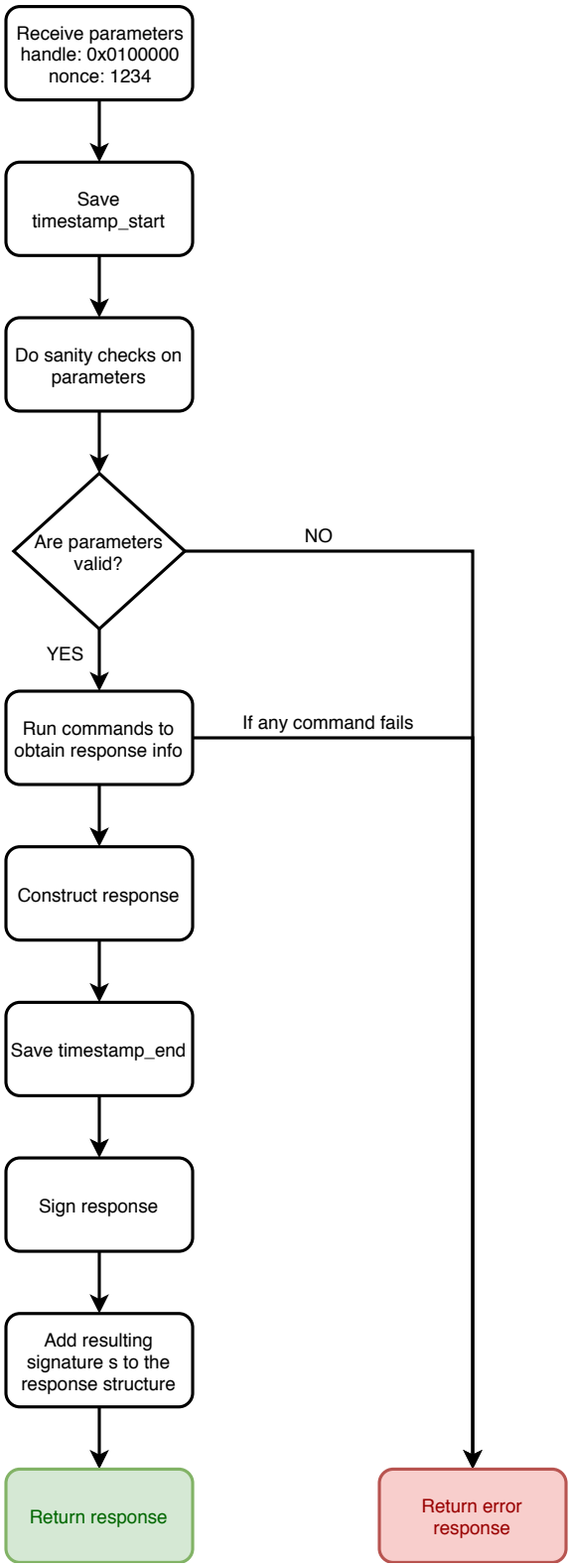

Figure 20: Behaviour of the trust agent upon receiving a request

header with a nonce, which is used to prevent replay attacks. Figure 20 shows the behaviour of the trust agent upon receiving a request. It will store a timestamp when it starts processing the request, verify the parameters received, run the necessary commands in the platform to obtain the information requested, construct a response, store a timestamp when it is done processing, sign the response and return the response with the signature to the requester.

Field	Description
nonce	Nonce received in the request header
command	Kind of information to request, e.g. identity, capabilities or quote
key_handle	Key handle on the TPM used to sign the response
timestamp_start	Indicates when the request was received by the trust agent
timestamp_end	Indicates when the processing of the request ended
signature	Signature of the response object by the key indicated in the request

Table 12: Basic fields in a trust agent response

All responses from the endpoint are JSON objects that include at least the fields described in Table 12. Depending on the endpoint, more fields are included in the response. Table 13 shows the extra fields included in responses from different endpoints.

Endpoint	Response Field	Description
identity	identity	Dictionary that contains the identity information, usually EK and AK
	extra_data	Dictionary that contains additional identity information (e.g. system and kernel information)
capabilities	capabilities	Dictionary that contains the capabilities reported by the TPM (includes fixed properties, variable properties, commands, algorithms, ecc curves and several TPM handles)
quote	quote	Dictionary that contains the quote structure returned by the TPM
	pcrs	Dictionary that contains the current value of the PCRs
keys	key	Contains the public key requested
get_ev	expected_value	Contains the expected value for the requested PCRs

Table 13: Extra fields in a trust agent response

5.2.2 Boot agent

The boot agent is a script implemented using Python, which reports a boot event to the attestation server whenever the element is powered on/off. The boot agent

is deployed as a oneshot systemd service. When the service is started, it reports a START boot event. Similarly, when the service is stopped, it reports a STOP boot event.

5.3 Provisioning tools

After adding the machine to be monitored by the attestation server, we need to provision it for using the machine agents. We store the configuration for the machine agents in the NVRAM of the TPM and seal them against a set of PCRs, so the agents can only be used when the platform is in a correct state. When we store information in the NVRAM of the TPM and seal it against a PCR, this information can only be read again if the PCR contains the same value as it had when the information was written.

The configuration stored in NVRAM for the trust agent includes the key handles for the EK and AK in the TPM and the hashing algorithm to be used by the TPM. For the boot agent, we store the ID of the element in the database.

Appendix F shows the code details for provisioning the machine. We wrote a Python library that does the provisioning for the element. Figure 21 shows the provisioning tools being run in one of our machines.

```
$ provision_machine -p sha256:0
Machine successfully provisioned.
Configuration:
TPM_EK_KEY_HANDLE = "0x81010002"
TPM_AK_KEY_HANDLE = "0x81010003"
TPM_HASH_ALGORITHM = "sha256"

NVRAM index: 0x1800005
Authorization hierarchy: 0x40000001
NVRAM sealed against PCRS: sha256:0

# compute @ compute in ~ [16:06:51]
$ provision_element_id -e 5b4894d419af5157a0f61260 -p sha256:0
Machine successfully provisioned.
Element ID: 5b4894d419af5157a0f61260
NVRAM index: 0x1800006
Authorization hierarchy: 0x40000001
NVRAM sealed against PCRS: sha256:0
```

Figure 21: Provisioning of an element

When sealing the configuration for the machine agents, any PCR can be used. One possibility is to seal the configuration of the agents against the measurement of the agent code. We can create a new SELinux type and use it to tag the agent files. Later, we can create a Linux IMA policy, which measures the files and stores the integrity value over those measurements in a specific PCR, e.g. PCR 14. During

machine provisioning, we can seal the agent configuration against PCR 14. If the code of any of the agents is modified, both the measurement for the file and the integrity value stored in PCR 14 will change.

This provisioning stage aims to prevent the situation in which an attacker would modify the code of the machine agents and provide false information to the attestation server. We seal the configuration needed for the agents to work properly against the measurement of the actual code of the agents. Therefore, if the code of the agents is modified in any way, these agents will no longer be able to retrieve their configuration in order to communicate with the attestation server.

5.4 Attestation server and database

The attestation database was implemented in MongoDB [13], and it is only accessible by using the attestation server as an intermediary. The attestation server is implemented as a Flask server using Python 2.7, which exposes a REST API. Table 14 shows the defined endpoints for the attestation server API.

The `/check_element_trust` is used to request the attestation server to evaluate the trust status of an element by running a ruleset against the element and a policy set. The HTTP request should include a JSON object with parameters for this action. The parameters are: ID of the element, ID of the ruleset, ID of the policy set and a boolean that indicates whether the attestation server should requote the element before running the ruleset or not.

The `/take_quote` endpoint is used to ask the attestation server to quote an element. The HTTP request should include a JSON object with parameters for this action. The parameters are: ID of the element and ID of the policy set. This will trigger interaction with the trust agent of the element to retrieve a fresh quote.

The `/elements` endpoint is used to obtain a list of known elements in the database and to add new elements. It can receive `GET` and `POST` requests. A `GET` request will retrieve all the elements known to the attestation server. A `POST` request includes a JSON object with the following parameters: name, openstack ID, IP address, TPM EK, TPM AK, system information, ID of default policy set, ID of default ruleset, a list of kind tags and trust status. The server response includes the ID of the element in the database.

The `/elements/(element_id)` endpoint is used to obtain details of a particular element, update the information of an element or delete an element from the database. It can receive `GET`, `PUT` and `DELETE` requests. A `GET` request will get the details of the element indicated in the URL. A `PUT` request includes a JSON object with the parameters and new values. A `DELETE` request will delete the record of this element

[13] https://www.mongodb.com/

URL	Operation	Description
/check_element_trust	POST	Runs a ruleset against a policy set
/take_quote	POST	Quotes an element against a policy set
/elements	GET, POST, PUT, DELETE	Used to manage elements in the database
/elements/(element_id)/capabilities	GET	Gets the capabilities for a specific element
/events	GET, POST	Used to obtain existing events or create new ones
/elements/(element_id)/events	GET	Gets the events for a specific element
/policies/(policy_id)/events	GET	Gets the events for a specific policy
/policy_sets/(policy_set_id)/events	GET	Gets the events for a specific policy set
/policies	GET, POST, PUT, DELETE	Used to manage policies in the database
/policy_sets	GET, POST, PUT, DELETE	Used to manage policy sets in the database
/quotes	GET, POST, DELETE	Used to manage quotes in the database
/elements/(element_id)/quotes	GET	Gets the quotes for a specific element
/rulesets	GET, POST, PUT, DELETE	Used to manage rulesets in the database

Table 14: Attestation server REST API endpoints

in the database.

The `/elements/(element_id)/capabilities` endpoint accepts `GET` requests. This endpoint will trigger an interaction with the trust agent of the element to obtain a capability report.

The `/events` endpoint is used to obtain a list of known events in the database and to

add new events. It can receive `GET` and `POST` requests. A `GET` request will retrieve all the events known to the attestation server. A `POST` request includes a JSON object with the necessary parameters to create an event in the database. The parameters in the request will depend on the type of event being created. Section 4.8 includes the fields that each kind of element must have. The server response includes the ID of the event in the database.

The `/events/(event_id)` endpoint accepts `GET` requests and is used to obtain details of a particular event. The `/elements/(element_id)/events` endpoint accepts `GET` requests and is used to obtain all events for a specific element. The `/policies/(policy_id)/events` endpoint accepts `GET` requests and is used to obtain all events for a specific policy. The `/policy_sets/(policy_set_id)/events` endpoint accepts `GET` requests and is used to obtain all events for a specific policy set.

The `/policies` endpoint is used to obtain a list of known policies in the database and to add new policies. It can receive `GET` and `POST` requests. A `GET` request will retrieve all the policies known to the attestation server. A `POST` request includes a JSON object with the following parameters: name, pcrs, expected value and kind. The server response includes the ID of the policy in the database.

The `/policies/(policy_id)` endpoint is used to obtain details of a particular policy, update the information of a policy or delete a policy from the database. It can receive `GET`, `PUT` and `DELETE` requests. A `GET` request will get the details of the policy indicated in the URL. A `PUT` request includes a JSON object with the parameters and new values. A `DELETE` request will delete the record of this policy in the database.

The `/policy_sets` endpoint is used to obtain a list of known policy sets in the database and to add new policy sets. It can receive `GET` and `POST` requests. A `GET` request will retrieve all the policy sets known to the attestation server. A `POST` request includes a JSON object with the following parameters: name, list of policy IDs and kind. The server response includes the ID of the policy set in the database.

The `/policy_sets/(policy_set_id)` endpoint is used to obtain details of a particular policy set, update the information of a policy set or delete a policy set from the database. It can receive `GET`, `PUT` and `DELETE` requests. A `GET` request will get the details of the policy set indicated in the URL. A `PUT` request includes a JSON object with the parameters and new values. A `DELETE` request will delete the record of this policy set in the database.

The `/quotes` endpoint is used to obtain a list of known quotes in the database and to add new quotes. It can receive `GET` and `POST` requests. A `GET` request will retrieve all the quotes known to the attestation server. A `POST` request includes a JSON object with the following parameters: ID of the element, kind, ID of the policy, quote and pcr values. The server response includes the ID of the quote in the database.

The `/quotes/(quote_id)` endpoint is used to obtain details of a particular quote or delete a quote from the database. It can receive `GET` and `DELETE` requests. A `GET` request will get the details of the quote indicated in the URL. A `DELETE` request will delete the record of this quote in the database. The `/elements/(element_id)/quotes` endpoint accepts `GET` requests and is used to obtain all quotes for a specific element.

The `/rulesets` endpoint is used to obtain a list of known rulesets in the database and to add new rulesets. It can receive `GET` and `POST` requests. A `GET` request will retrieve all the rulesets known to the attestation server. A `POST` request includes a JSON object with the following parameters: name, name of the python class that implements the rule, list of rules it depends on and kind. The server response includes the ID of the ruleset in the database.

The `/rulesets/(ruleset_id)` endpoint is used to obtain details of a particular ruleset, update the information of a ruleset or delete a ruleset from the database. It can receive `GET`, `PUT` and `DELETE` requests. A `GET` request will get the details of the ruleset indicated in the URL. A `PUT` request includes a JSON object with the parameters and new values. A `DELETE` request will delete the record of this ruleset in the database.

5.5 Attestation libraries

The attestation server exposes a REST API that can be used to retrieve information from it. We have implemented a set of attestation libraries in Python 2.7. These libraries manage the HTTP request interactions with the attestation server and perform error handling. This abstracts away the HTTP layer from the user, which can now communicate with the attestation server by using python functions.

For each item in the attestation database, we define an item manager, which implements functionality such as creation, deletion, update and retrieval of the items from/to/in the database. The manager knows how to communicate with the attestation database and abstracts those details away from the user of the libraries.

Additionally, we define a Python class representation of each type of item in our database. Therefore, when using the attestation libraries, we can deal with objects instead of plain dictionaries or JSON structures. Figure 22 shows an example interaction using the attestation libraries. In this case, the user wants to obtain all the quotes in the attestation server. It creates a `QuoteManager` instance, and uses the `get_quotes()` function. The attestation libraries will abstract away the request exchange with the attestation server and return an instance of the class `QuoteList`, which contains instances of the `Quote` class.

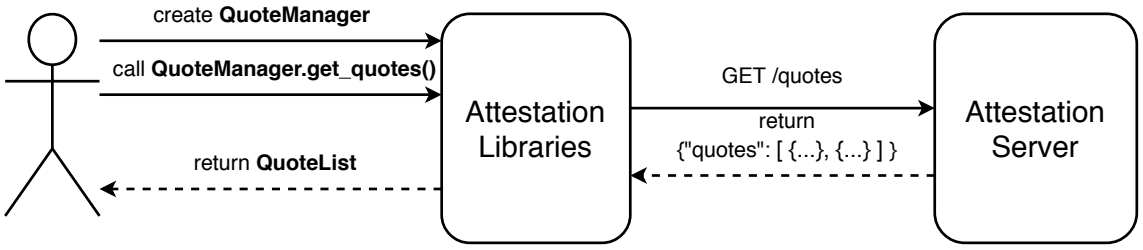

Figure 22: Example of attestation library usage

5.6 Attestation UI

The attestation UI provides a web dashboard for the system administrator. It shows an overview of the trust status of the elements in the cloud. Additionally, it can interact with the attestation server to trigger quoting or trust checks. It is implemented as a Flask server using Python 2.7. In the UI, we can find different pages for each relevant part of the system. This section describes the most relevant views of the attestation UI.

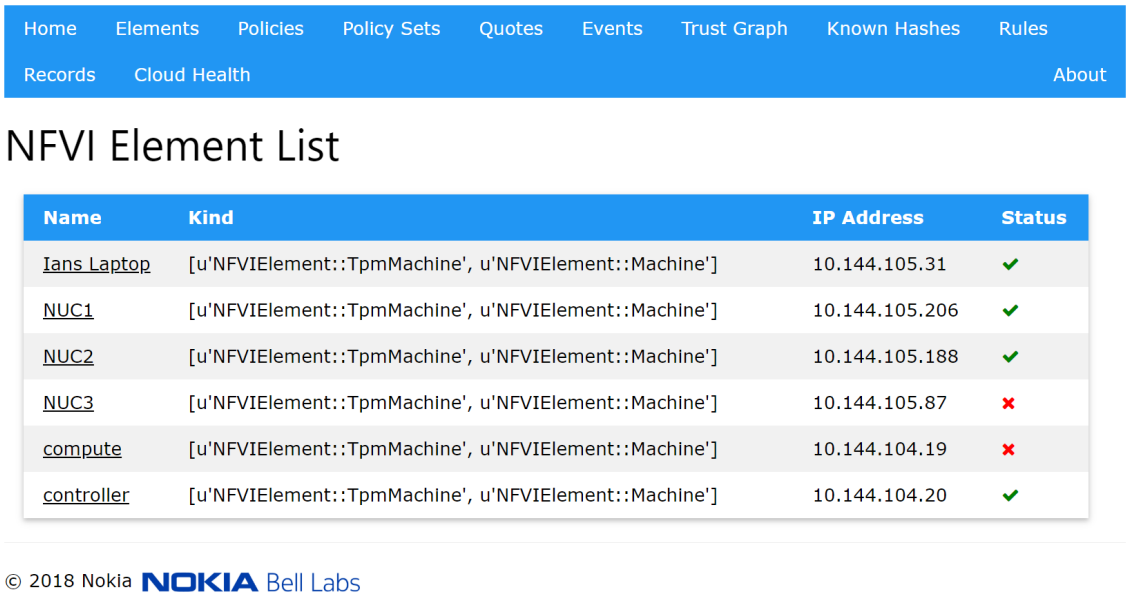

Figure 23: Attestation UI element view

Figure 23 shows the elements view of the attestation UI. This view includes details on all the elements monitored by the attestation server as well as the trust status. When any of these elements is clicked, it displays a new page with all the element details known by the attestation server. From that page, it is possible to trigger a quote or trust check for the element, view the events or quotes for that element and view a monitoring page with details of the trust status of the element over time.

The UI also provides a view with the details for the policies and policy sets known

to the attestation server. Another view is dedicated to all the quotes stored by the attestation server for all elements monitored. Similarly, we include a view, which includes the events reported to the attestation server. On the next view, we find information about the rules used in the attestation server for trust checks. Finally, the cloud health view provides a quick visual overview of the trust status of the cloud, by using pie charts that indicate the amount of trusted and untrusted elements in the cloud.

5.7 Use cases

In this section, we examine a set of important use cases for our system. We describe how to add elements to be monitored by the attestation server, how to obtain measurements from this element, how to check the trust status of an element and, finally, how to use our RCA mechanism to investigate attestation failures.

5.7.1 Introduce new element for monitoring

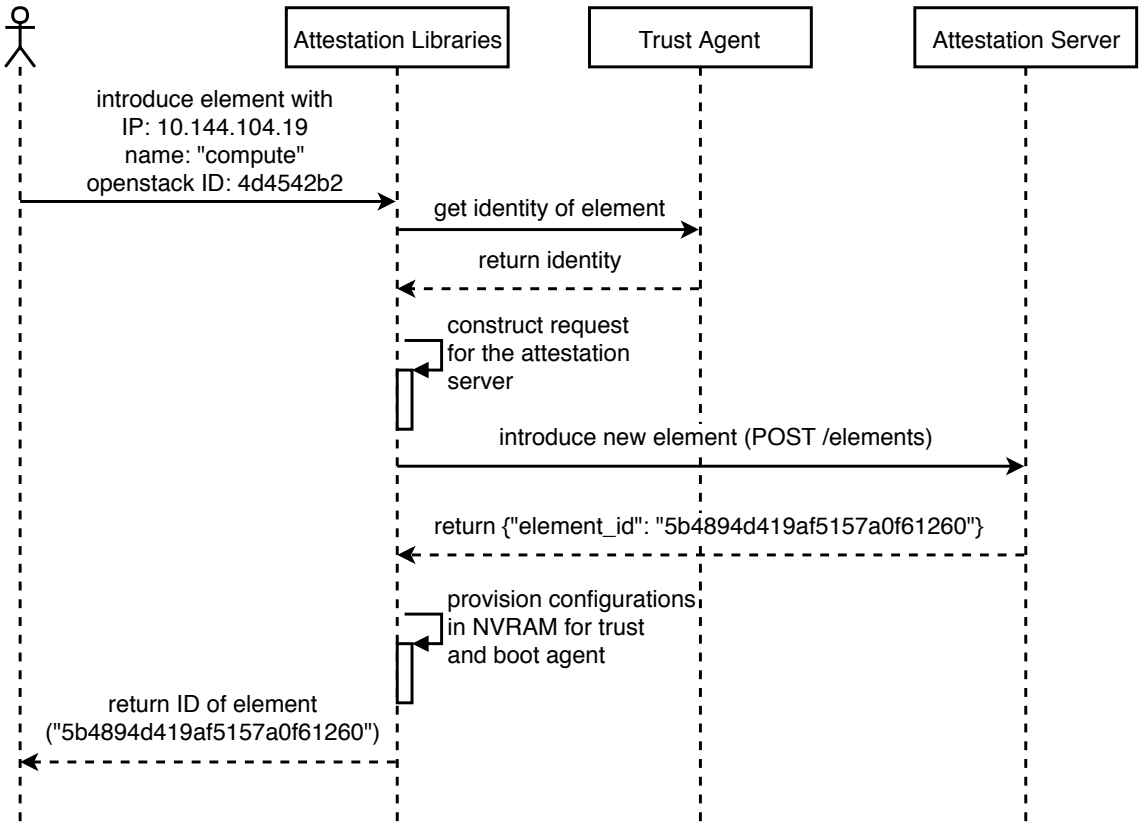

Figure 24: Interaction between attestation components to introduce a new element

The attestation libraries include a tool for introducing a new element to be monitored

by the attestation server. Figure 24 describes the interaction between the administrator, the attestation library, the trust agent in the element and the attestation server when introducing a new machine. The administrator uses the tool provided by the attestation libraries to introduce a machine with IP address "10.144.104.19", name "compute" and openstack ID "4d4542b2". Then, the attestation libraries interact with the `/identity` endpoint of the trust agent to obtain the identifying information of the element (EK, AK and system information). With all the information about the element, the attestation libraries generate an HTTP POST request to the attestation server to create a new element.

```
{
  "name": "compute",
  "openstack_id": "4d4542b2",
  "ip": "10.144.104.19",
  "ek": "-----BEGIN PUBLIC KEY-----\
      nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAh6V2QY/i15sdHj+U4Y9r\
      nWSUU6jejZPScL+8K0LWrCxQmEycLGTNMwXw4f0TabeIKPpY2BcmWRKWJ7c2a/b4v\
      np7exqgYGBru0B//qdrZvcth1OKEZUsrIFYGvORjUfaSAoMiB4LQkqpEf/1jxO/zI\
      n6EanTHm1oN/qqHh2AkKHJUJ2rQUobwE+2wxEmvnW7AVD67xliBfGczC9LDbwDOag\
      nDalbLRLeiFt1qTj23tDhJGjtYqKvLMBTp4mzBukAqnuEEU+GwX8/Bdy7NPGQE/GV\
      nXvfB0q6XHHgn8EQSNTNszPUwMhPRgbL3mJ8NWoQ+BuNSdnifC7+M3WCfI0TVUePb\n/QIDAQAB
      \n-----END PUBLIC KEY-----\n\n",
  "ak": "-----BEGIN PUBLIC KEY-----\
      nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmZtDWjzQFuZPiqkDLOJh\
      nF3kWsu0pXT1zzjwDkbMBYtVnZB99c4w8afG5hQWNswqaDg/tikB55vJ7tS94tPM8\nT/
      CvWt0qLoR0z8Pg1o+V2WcTJEnqYi/X9Rs0e9jNNRzrp40LquRR6BCJIwt9tDvW\
      n4GnU2AEDpRDodUGYUsN4tN84sYLesDKCZcVjEInxBidWoA4CUPdJ3NexZAIgYRFs\
      nXgi3joPTYne2ySKhKpTV8g7rkO9Jjkfd7EEO0vvPx4aQ2ke0tWBGDi+HwTrBMOzS\
      nNRKo5mNnS32H9cl0yeaC6qkAio2LvbwhJMELlAAOmgmAAEc4P0fSwgqZJDkXfhUm\nmwIDAQAB
      \n-----END PUBLIC KEY-----\n\n",
  "uname": "Linux compute 4.10.0-42-generic #46-Ubuntu SMP Mon Dec 4 14:38:01 UTC
       2017 x86_64 x86_64 x86_64 GNU/Linux",
  "policies": [],
  "kinds": ["NFVIElement::TpmMachine"],
  "status": False,
  "ruleset_id": null,
}
```

Listing 1: POST request parameters to create a new element

Listing 1 shows the body of the POST request sent to the attestation server. The attestation server adds the new element to the database and returns the ID of the element. The attestation libraries will then run the provisioning functions for the element. After the element has been provisioned, the tools return the element ID to the administrator. If the element introduction was successful, the element now appears in the attestation UI, as shown in Figure 25.

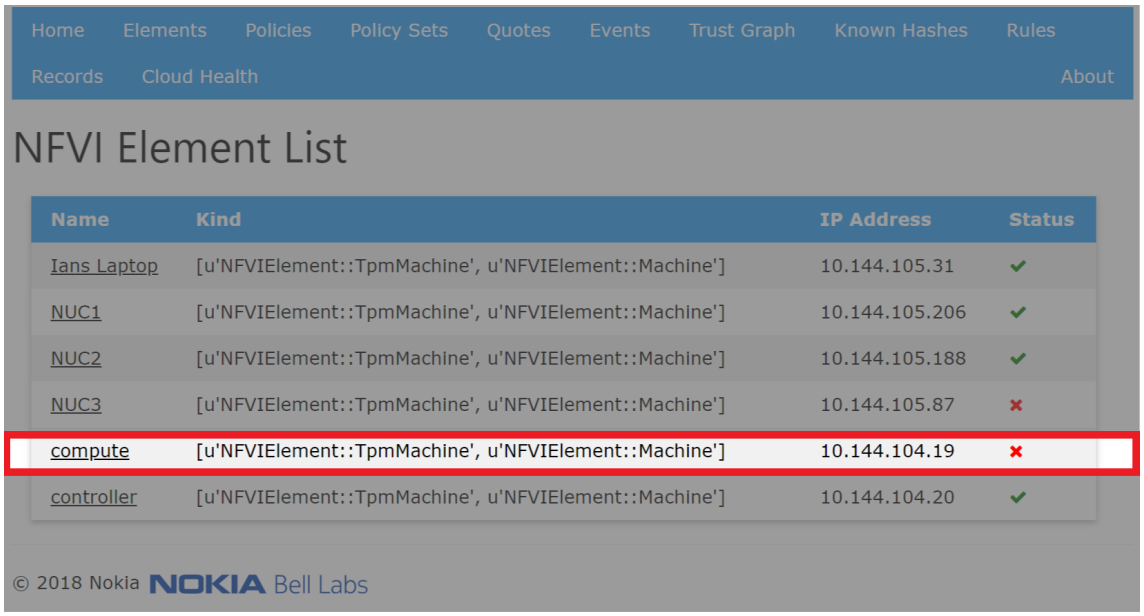

Figure 25: Attestation UI after introducing a new element

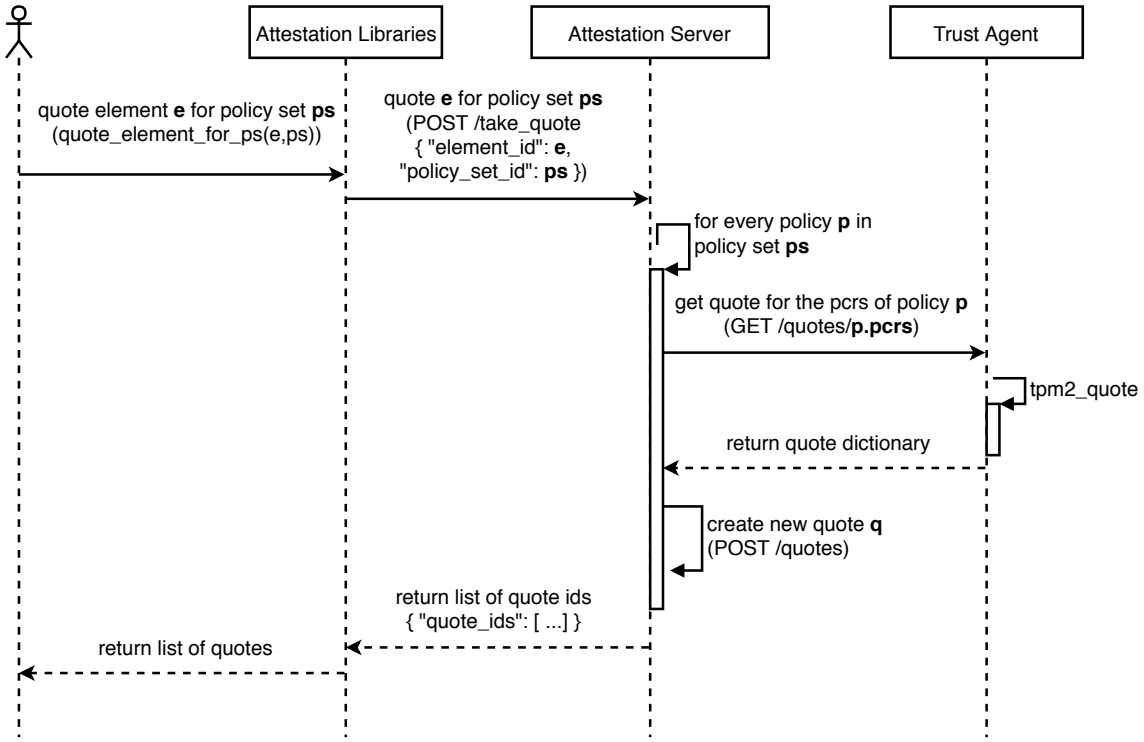

Figure 26: Interaction between attestation components to quote an element

5.7.2 Quoting an element

The attestation libraries include a tool for quoting an element monitored by the attestation server. Figure 26 describes the interaction between a user, the attestation

library, the trust agent in the element and the attestation server when quoting a machine. The user that wants to trigger a quote uses the function provided by the attestation libraries to quote an element for a policy set, if the policy set is not given the default policy set stored for the element in the database is used. The attestation library requests the attestation server to take quotes of the target element. Then, the attestation server requests a quote from the trust agent for every policy in the policy set. The attestation server accumulates a list of quote IDs and returns this list to the attestation libraries. Finally, the attestation libraries return a list of quotes to the user.

For convenience, the attestation UI provides the same mechanism. Figures 27, 28 and 29 show the quoting process triggered from the attestation UI. The user goes to the page of the element, chooses a policy set and clicks the “Quote” button. The process described earlier happens in the background. Finally, the UI prints a message with links to the new quotes for the element.

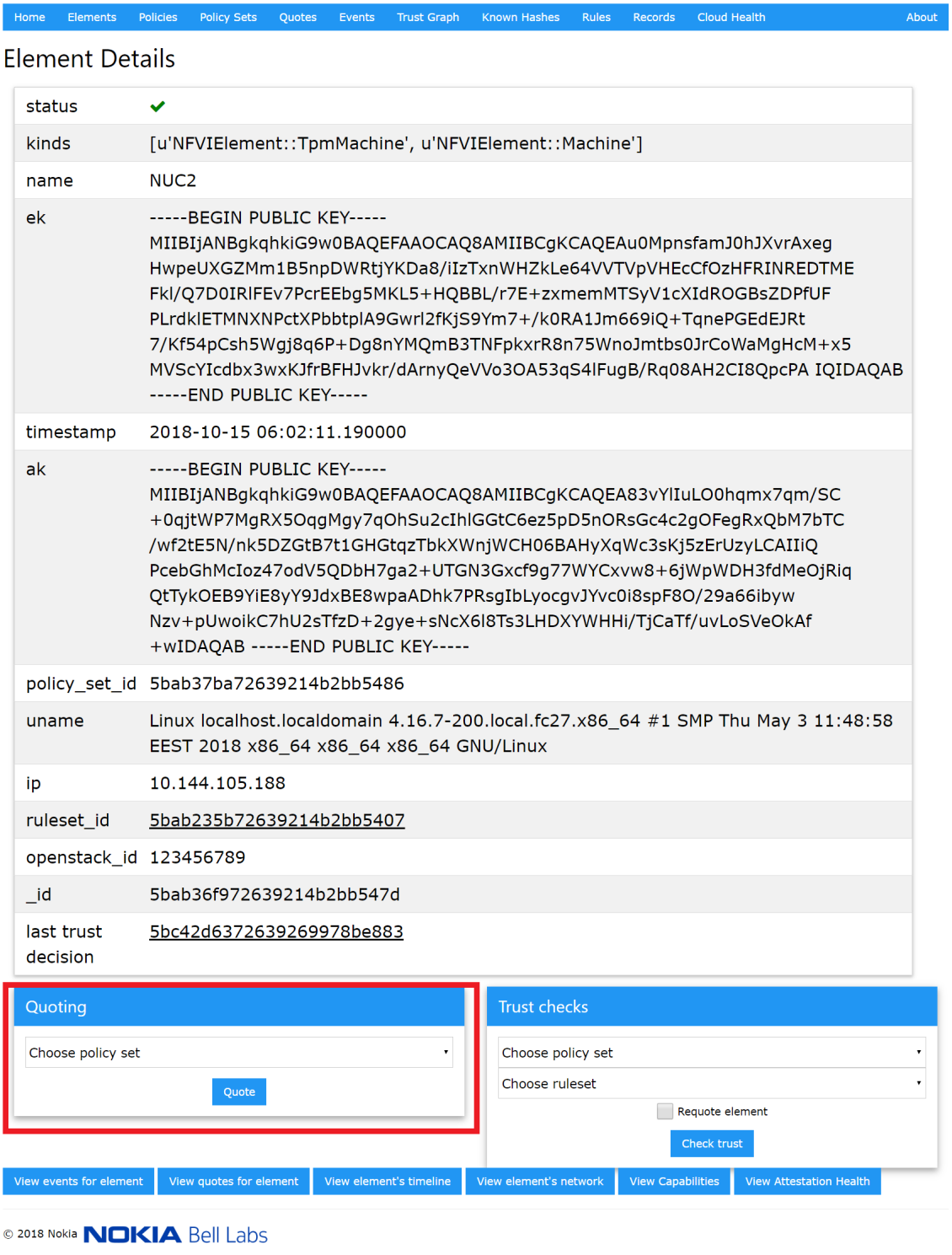

Figure 27: Quoting via Attestation UI (Step 1: Choose a policy set)

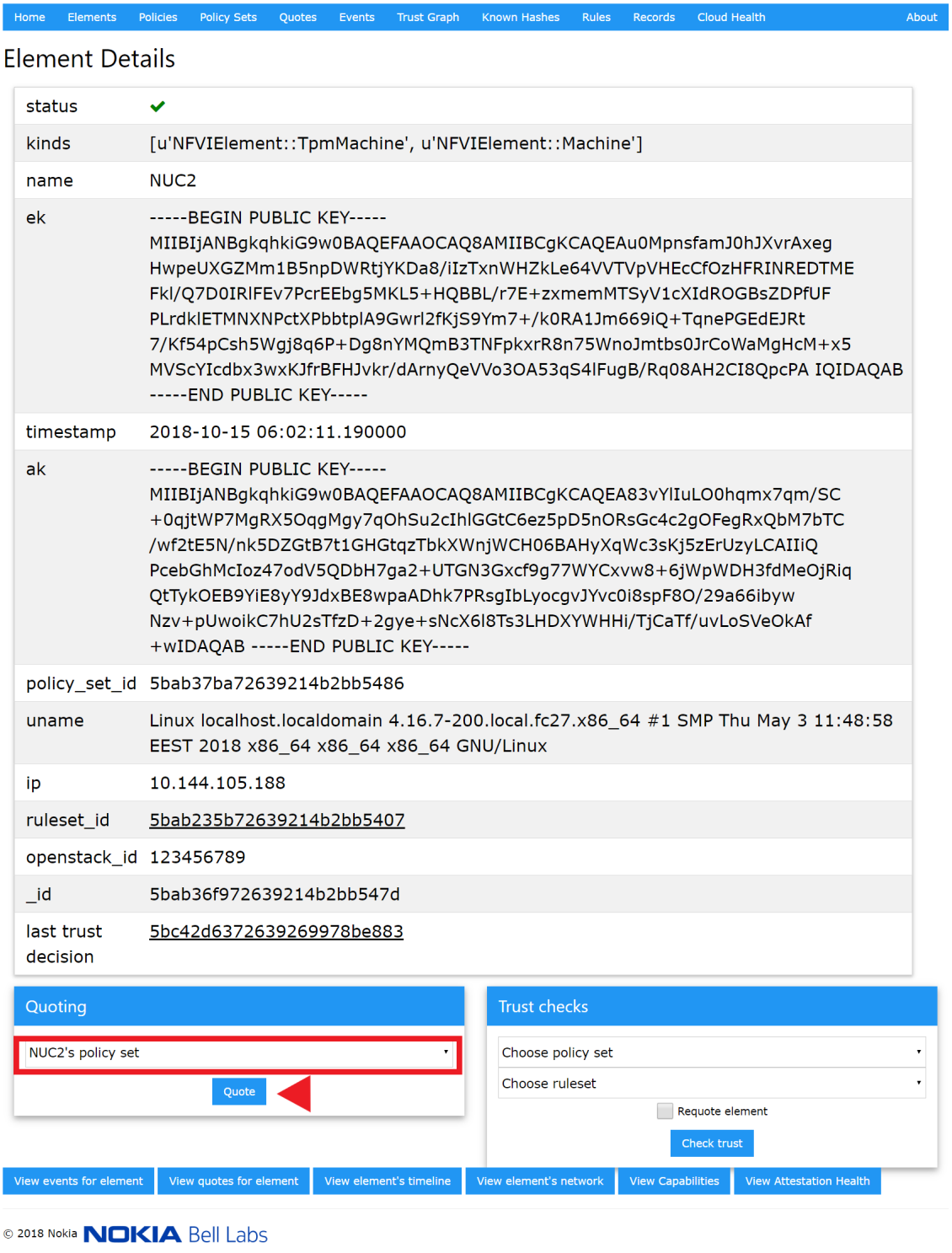

Figure 28: Quoting via Attestation UI (Step 2: Click “Quote” button)

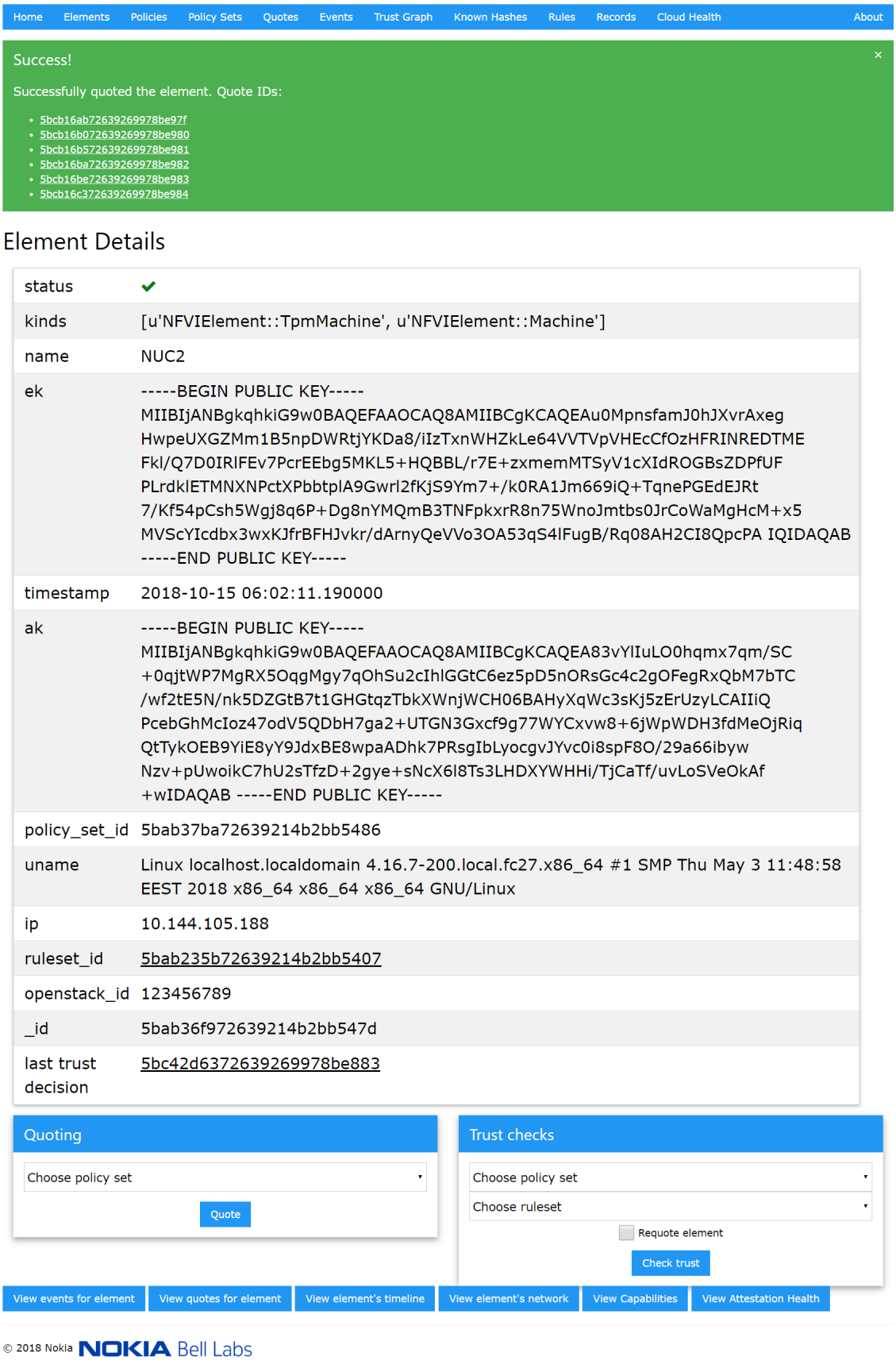

Figure 29: Quoting via Attestation UI (Step 3: Successfully quoted element)

5.7.3 Check element Trtst

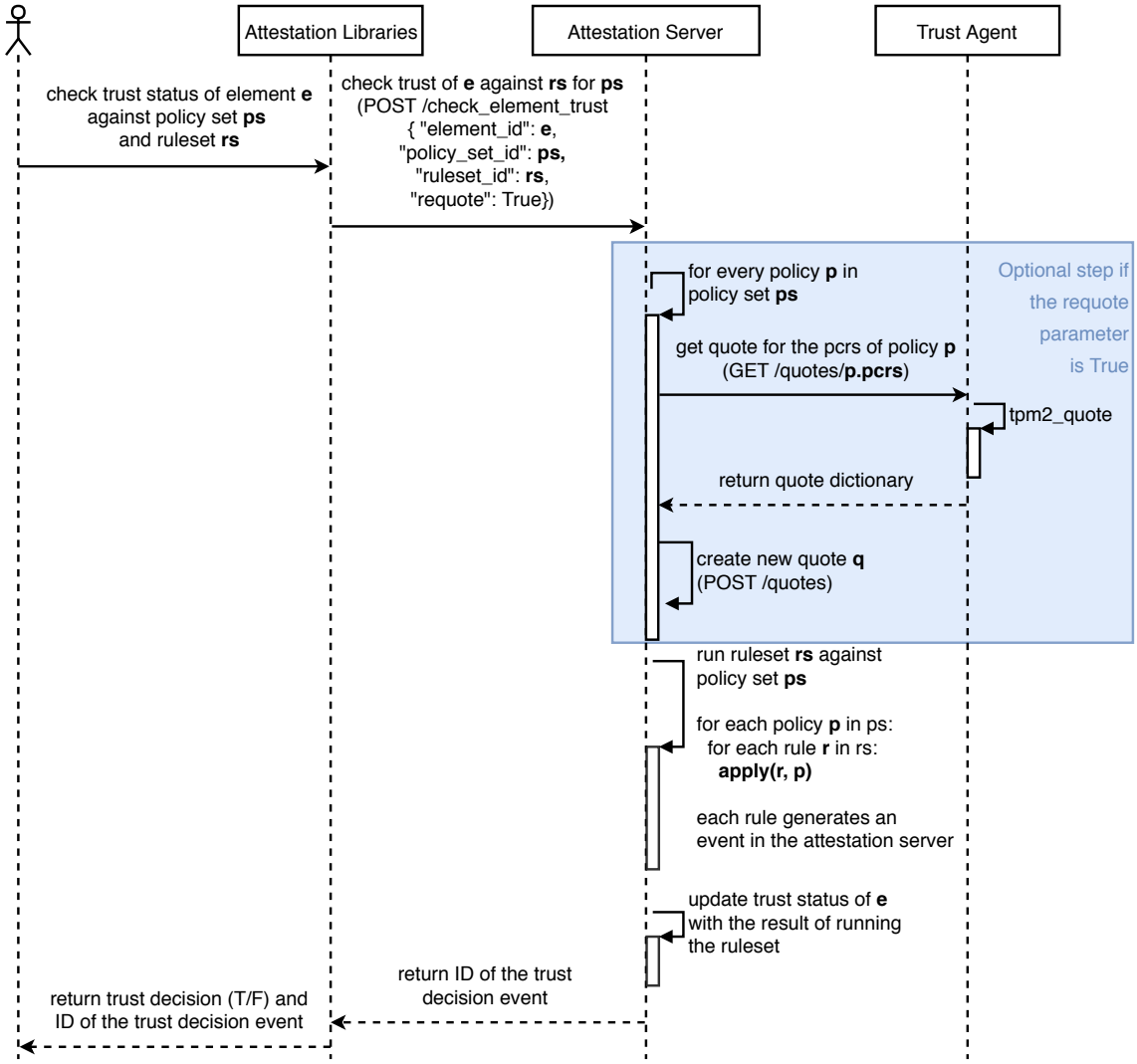

Figure 30: Interaction between attestation components to check the trust of an element

Machine trust is checked by running a ruleset against an element and a policy set. The attestation libraries include a tool for triggering a trust check of an element monitored by the attestation server. Figure 30 describes the interaction between a user, the attestation library, the trust agent in the element and the attestation server when checking the trust of an element. The user wants to check the trust status of an element e according to a ruleset rs and a policy set ps, so he uses the attestation libraries to trigger a trust check in the attestation server. The attestation server receives a POST request with the ID of the element, ID of the policy set, ID of the ruleset and a parameter that indicates whether fresh quotes should be taken. If the ID of the policy set or ruleset is not included in the parameters, the attestation server will use the default policy set or ruleset for the element.

If the requote parameter was set in the request, the attestation server takes quotes of element e for all policies in the policy set. Then, the attestation server will evaluate the ruleset against the policy set. For every policy set, it applies every rule in the ruleset, as described in Chapter 4, Section 4.6. Each time a rule is applied, an event is generated in the attestation server. After evaluating the trust of the element, the attestation server updates the trust status of the element in the database. The final trust decision will be stored as an event in the attestation server and the attestation server will return the ID of this event to the libraries. The libraries will return the trust decision and the ID of the event to the user. For convenience, the attestation UI provides the same mechanism.

5.7.4 Analyzing attestation failures

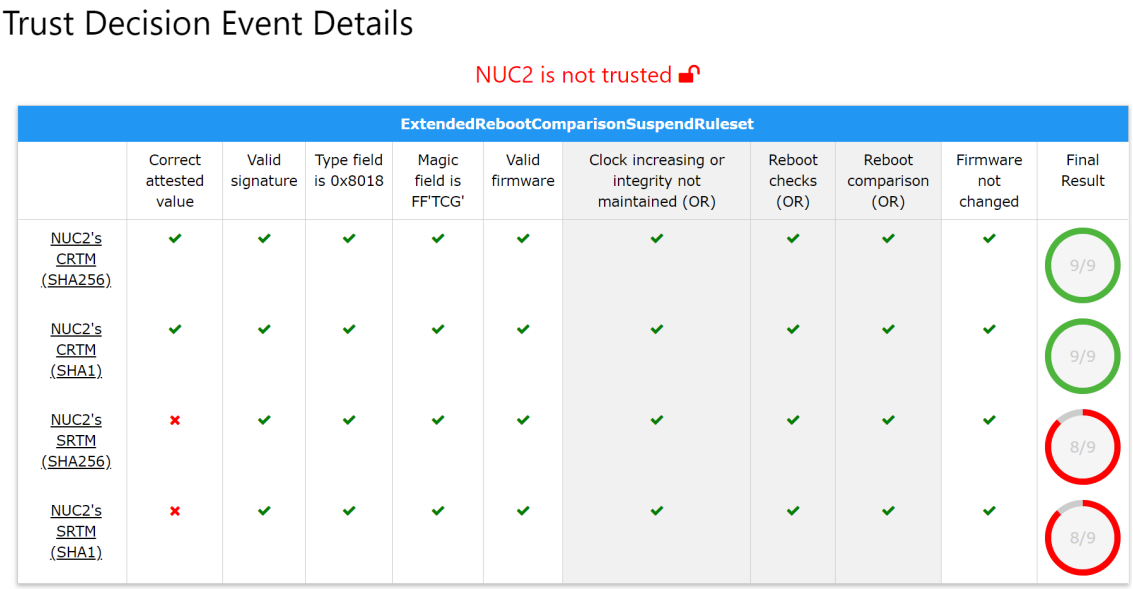

Figure 31: Attestation UI summary of a trust decision

In this section, we describe how an attestation failure is discovered and further analyzed in the system. Figure 31 shows the detailed view provided by the attestation UI after the trust status of an element is evaluated according a policy set and ruleset, as described in Section 5.7.3. This summary view allows us to quickly identify what rule in the ruleset failed. In this case, the failure was due to a mismatch between the measurements and the expected value for the policy.

Figure 32 shows the CFT to determine the root cause of the failure and possible mitigations. In the attestation UI, we can access the two latest quotes used to determine platform trust, as shown in Figure 33. We can see that the 'attested' field changed between quotes. By following the causes of this tree, we determine that the attested value changed between quotes, and, by checking the element log, we see that the element has not been updated. Therefore, the element must have received an

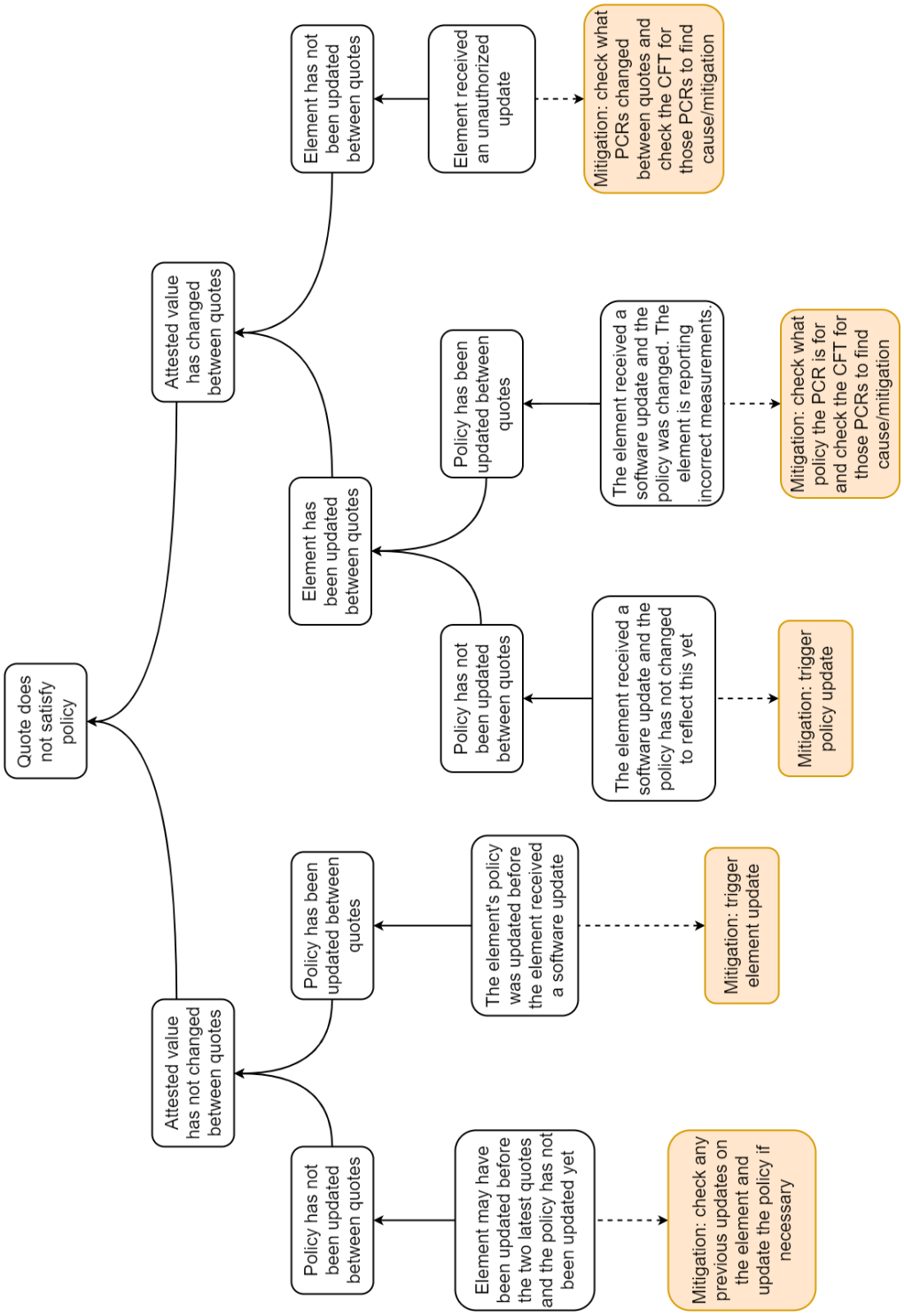

Figure 32: CFT: Quote does not satisfy policy

unauthorized update. The mitigation is to check a second CFT for specific actions depending on what PCRs changed.

Figure 35 shows the PCRs for the two latest quotes. We can see that PCR 1 changed,

Quote

Attested Value	l5L7/BsERIOQisRxPNUDNwA07Rs0XTzhN9/KhgQ1g0Y=
TPM Clock	8082294754
Extra Data	c/UOdgWeT12hM9e+X4OPZg==
TPM Firmware Version	1407641172325888
Magic	/1RDRw==
Qualified Signer	toah8XFBJ5KH+C1dBe2O4rrJcA5QZGZx+q7r4jkYqoU=
Quote File (base64 encoded)	/1RDR4AYACIAC7aGofFxQSeSh/gtXQXtjuK6yXAOUGRm
Reset Count	180
Restart Count	0
Safe	1
Signature	ABQACwEAhweKaKKT5YNQlddqU9LCcfBcQ4sd0Pv13gaY
Type	gBg=

(a) First quote

Quote

Attested Value	BOqpSWCwFCtWm0NBy36HYPNUVYpWEwQZTffx7iPSHiw=
TPM Clock	11231493591
Extra Data	pvUFuF6QTyK15wx2QIwj+g==
TPM Firmware Version	1407641172325888
Magic	/1RDRw==
Qualified Signer	toah8XFBJ5KH+C1dBe2O4rrJcA5QZGZx+q7r4jkYqoU=
Quote File (base64 encoded)	/1RDR4AYACIAC7aGofFxQSeSh/gtXQXtjuK6yXAOUGRmcfq
Reset Count	181
Restart Count	0
Safe	1
Signature	ABQACwEA8NOFaE4PbHiFBDFXkO7M4bL+scFMN/F3vN1a9
Type	gBg=

(b) Second quote

Figure 33: Latest quotes for NUC2

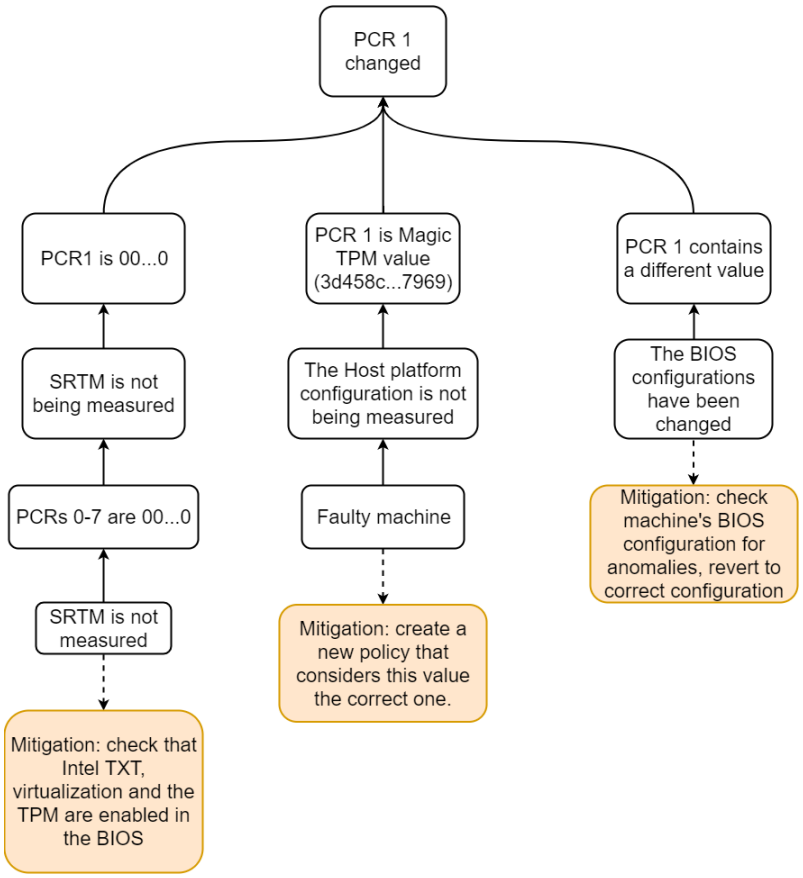

Figure 34: CFT: PCR 1 changed

so we check the CFT for changes on PCR 1 (Figure 34). From this tree, we determine that the root cause of the problem is that the BIOS configurations have been changed. Then, as a mitigation, we can check the BIOS configuration of the affected machine and revert the settings to the correct values.

5.8 Summary

In this chapter, we presented the implementation details of the different components of the remote attestation solution introduced in Chapter 4. Additionally, we explore in detail different use cases in the system.

sha256	0	33ba0f3c29f478c53fa626b13b2dbed2717c90b000aaaf3a3990b233b731c5fc
	1	386a38e82c8d393fb3e59e8e9a0e797832916af0122ef316781883f01a94980e
	2	b4b9d9f8c4a61762ef5652a8536e32ce4b4c027bcae4fda787a38e7eedaa2f81
	3	3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
	4	3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
	5	3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
	6	3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
	7	3d6207f9a2c3fa1db729f06e71b09d2e7ca7c0c198f6c1410c2186bbe2cc1826

(a) PCRs for the first quote

sha256	0	33ba0f3c29f478c53fa626b13b2dbed2717c90b000aaaf3a3990b233b731c5fc
	1	0d2416be35c1dc33d4b3860d218b71f296e4d900075583eb816c24da0a6974db
	2	b4b9d9f8c4a61762ef5652a8536e32ce4b4c027bcae4fda787a38e7eedaa2f81
	3	3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
	4	3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
	5	3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
	6	3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
	7	3d6207f9a2c3fa1db729f06e71b09d2e7ca7c0c198f6c1410c2186bbe2cc1826

(b) PCRs for the second quote

Figure 35: PCRs for the latest quotes for NUC2

6 Discussion and Results

This chapter presents an evaluation of the solution designed and implemented in this thesis using the identified problems in research from Chapter 3 as our evaluation criteria. Additionally, we evaluate the performance of the system and discuss changes that can be done to improve it. Finally, we compare remote attestation to whitelisting systems.

6.1 Evaluation

This section evaluates the implementation of this solution with regards to the problems in existing research identified in Chapter 3.

Run-time Integrity We address this problem by providing a mechanism that allows us to monitor critical files in the filesystem and detect if they have been modified. Furthermore, our proof of concept includes sealing the configurations for a component to a specific platform state. We implemented this option for securing the code of the trust agent in this work. This idea can easily be extended to protect the configuration files needed for OpenStack or the hypervisor running on the NFVI element. Therefore, these components will only work properly if their configuration has not been tampered with. The author in [55] includes an example of how this mechanism can be used to achieve trust in NFV MANO elements by sealing their configurations to a PCR value that stores the measurements for those configuration files.

In our approach, we utilize Linux IMA to measure specific files in our system and store an integrity value over these measurements in the TPM. We measure files that should not change during the lifecycle of the device. A change in the monitored files is detected when the integrity value stored in the PCR differs from the expected value. An enhancement to this approach would be to let the trust agent report to the attestation server the list of file measurements. Therefore, we could detect specifically which file was changed. In our thesis, the attestation server runs as a systemd service. However, the list of measurements taken by IMA are kept in the Linux securityfs[14], which is only accessible by root. We decided not to implement this functionality for security reasons, since it would require our trust agent to run with root privileges to be able to retrieve the IMA measurement list, thus opening an attack vector for privilege escalation on our devices.

Trustworthiness history We have created a system, which keeps a historic log of both the measurements taken for a platform and the trust decisions taken over

[14] https://lwn.net/Articles/153366/

those measurements. Furthermore, we provide a dashboard for the administrator to easily view how the trust status of the system has changed during its lifecycle. This dashboard includes information about the trust status over time and relevant fields of the quote, such as the number of reboots and the clock value. Figure 36 shows the attestation health dashboard for a specific machine in the system. This view of the attestation UI allows us to examine how the trust has evolved over time.

Limited definition of trust We have defined a rule system comprising a set of rules defined over the different fields of a quote. These rules utilize the policies defined in the attestation server to determine the correctness of the measurements obtained. Additionally, they may evaluate the changes in other parts of the quote, to detect any deviation from what is considered normal behaviour. These rules are combined into rulesets, which represent the definition of trust for an element.

Furthermore, these rulesets are extensible, which means that new rules can be created to handle new situations. As a result, we obtain a system that can easily adapt to new definitions of trust. This is particularly useful when we have machines that appear to misbehave, but do so in an expected way, as described in Chapter 4, Section 4.7.1. The resources provided by these faulty machines can still be leveraged, since the system allows us to introduce new rules to manage these situations.

Considering other systems Our attestation server is designed to record information about different events happening in the system. This mechanism is designed in a generic way, so the attestation server can track events that originate from a monitored device, as well as events originating from different components in the system, including NFV MANO components, path management systems and intrusion detection systems. At the proof of concept level, we evaluated the use of update and reboot events reported by the trust agent in the monitored device. The rulesets used to define trust for an element can consider these events when evaluating platform trust, as described in Chapter 4, Section 4.6. Although, in this work, we only used events reported by the devices themselves, the event system was designed to be extensible, so it can handle events from external systems.

Platform resilience The rule system introduced in this thesis is composed of rulesets. These rulesets represent differnt levels of trust depening on the rules and the amount of rules included in each ruleset. Furthermore, we can define a partial order over these rulesets based on their strictness, which provides us with a trust hierarchy.

Each trust decision for an element in the system is made against a ruleset and a policy set. Therefore, both current and previous trust statuses for a machine can always be linked to a level of trust in the hierarchy. This allows any application that

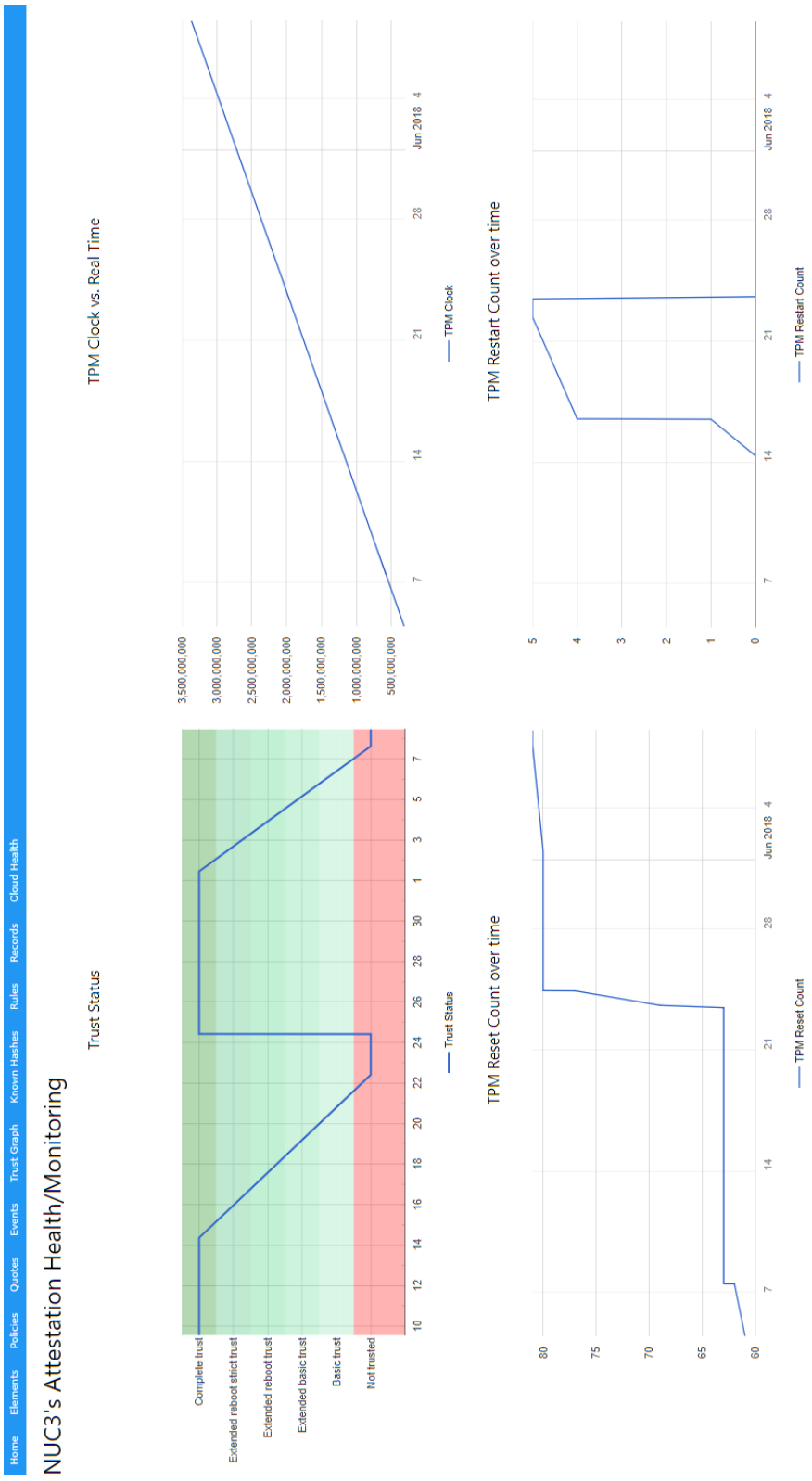

Figure 36: Attestation health dashboard in the attestation UI

uses the attestation results from the attestation server to determine how trusted their machines are based on the trust hierarchy defined by the rulesets. Depending on the level of trust, the application can decide what they allow the machine to do. For example, OpenStack, based on the trust level of a machine, can decide if it is enough to allow it to run a specific virtual workload. In addition, if the trust level is not acceptable, the attestation server allows triggering a new trust decision evaluation based on a stricter ruleset.

Analysis of attestation failures Finally, by using RCA techniques, we have defined a set of diagrams (CFTs and Ishikawa), which can be used to identify the causes of known system failures. We summarized the identified known failures and their causes in 11 CFTs and 8 Ishikawa diagrams at the proof of concept level. These diagrams can be used as tools to understand system behaviour after failures. Furthermore, they can be mapped to mitigations or actions to be taken when a failure is identified.

As far as we know, this is the first set of tools created to aid in computer forensics for attestation failures. This work introduces the diagrams as tools to be used by an administrator, and it is the first step towards developing a complete framework for attestation forensics. In the future, the analysis of attestation failures could be automated; however, this automation is considered out of scope for this thesis.

6.1.1 Performance evaluation

In this section, we evaluate the proposed solution in terms of performance. We measured the time taken to evaluate the trust for an element in the system. To evaluate the trust of an element, we requoted the element for fresh measurements and ran a ruleset. We tested all the rulesets defined in this work against an element in the system. This element was associated with a policy set containing four policies; therefore, each time we requote, we request four separate quotes from the trust agent on the device: one for each policy.

The process of evaluating the trust of an element comprises three main steps:

1. Requoting the element.

2. Running the ruleset.

3. Interacting with the attestation server (obtaining element and policy information, storing quotes, reporting trust decision events, etc)

Figure 37 shows the measurement results. We include a time breakdown by each step in the trust evaluation process, so we can evaluate which step takes the most time. Similarly, Table 15 contains the measurements taken.

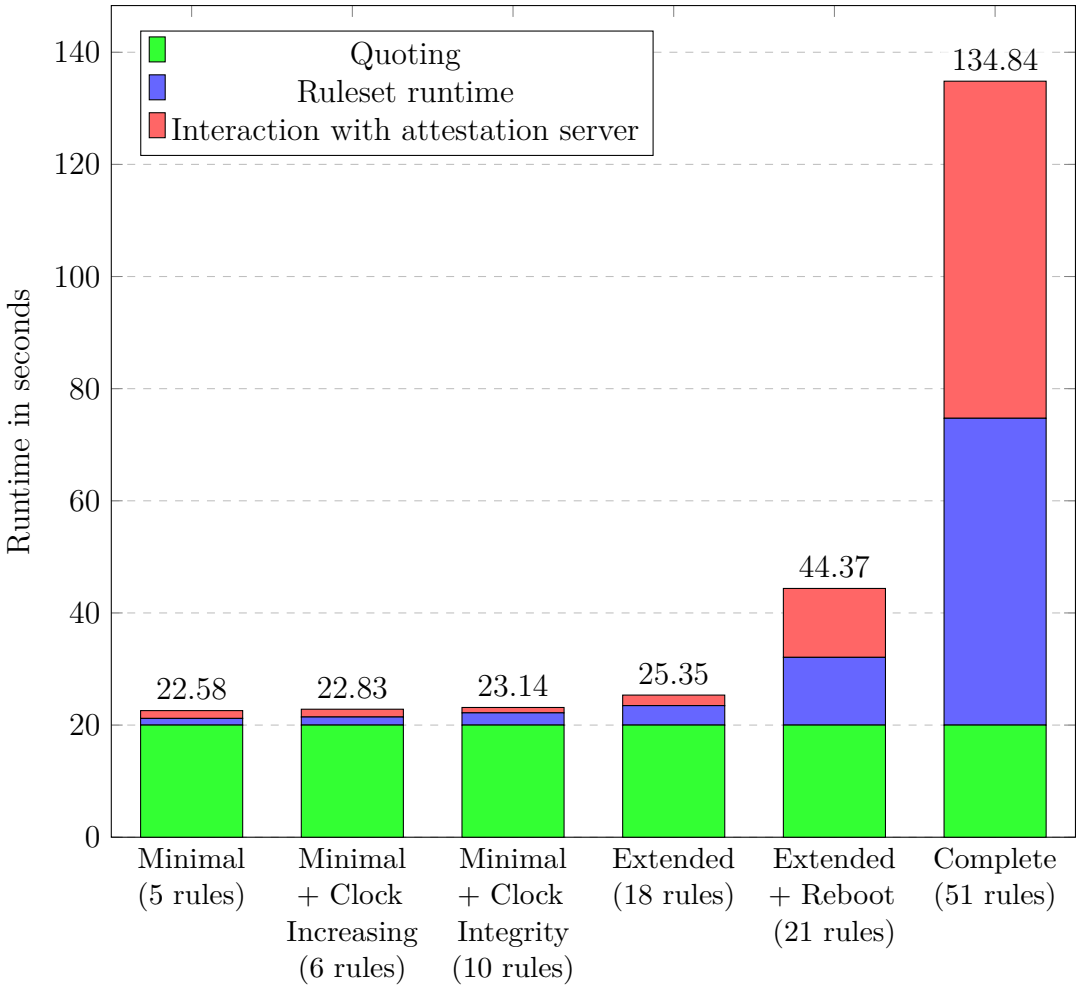

Figure 37: Performance evaluation: running rulesets

We can see that, for rulesets which contain less than 20 rules, requoting the element represents most of the time used for evaluating trust. Therefore, we decided to analyze the quoting process, to discover what operations took the longest time to run. When a trust agent receives a request for a quote, it must run a set of commands to obtain the necessary information to build the response. The main steps for building a quote response from the trust agent are:

1. Taking the quote (tpm2_quote).

2. Obtaining the current PCR values (tpm2_pcrlist).

3. Hashing and signing the response of the trust agent with a key from the TPM (tpm2_hash and tpm2_sign)

4. Extra processing of the data in the trust agent (receiving a request, building a response, etc.)

Ruleset	Number of rules	Total Runtime	Runtime Breakdown		
			Quoting	Ruleset runtime	Interaction with AS
Minimal	5 s	22.58 s	20 s	1.19 s	1.39 s
Minimal + Clock Increasing	6 s	22.83 s	20 s	1.44 s	1.39 s
Minimal + Clock Integrity	10 s	23.14 s	20 s	2.18 s	0.96 s
Extended	18 s	25.35 s	20 s	3.47 s	1.88 s
Extended + Reboot Comparison	21 s	44.37 s	20 s	12.08 s	12.29 s
Complete	51 s	134.84 s	20 s	54.72 s	60.12 s

Table 15: Runtime for each ruleset

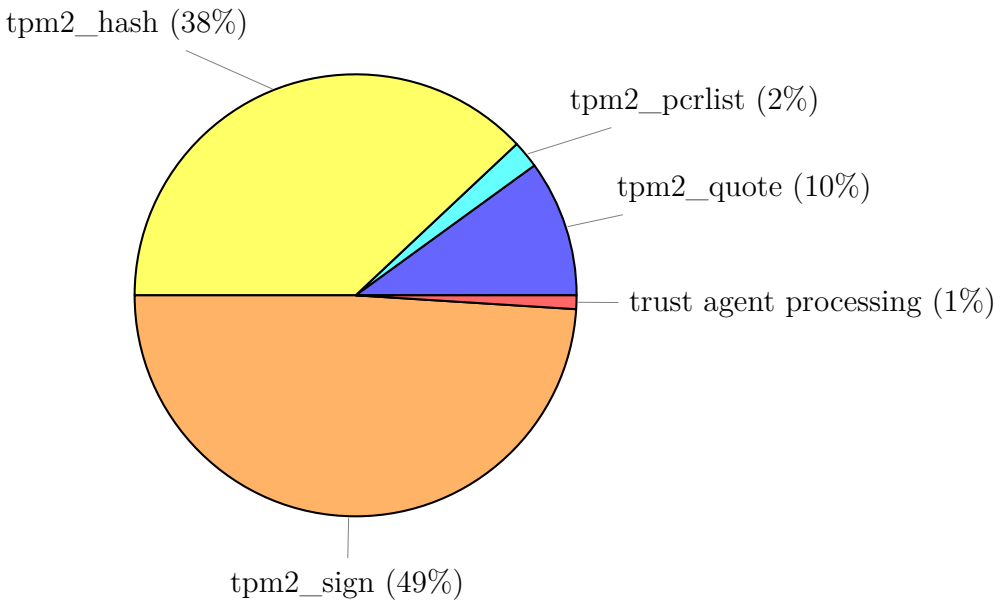

Figure 38: Performance evaluation: obtain quote from trust agent (time breakdown)

Figure 38 shows the time breakdown for obtaining a single quote for an element from the trust agent. 87% of the time used to obtain a quote is used for hashing and signing the response from the trust agent. Cryptographic operations on the TPM are slow, since it is not designed to be a cryptoaccelerator.

In this situation, we have a tradeoff between performance and security. To improve the performance of the trust agent, we could omit the signing of the complete response

sent by the trust agent and rely solely on the signature for the TPM2 quote structure. However, we could no longer guarantee that other information in the response, such as the PCR registers, reported by the trust agent have not been tampered with. At the proof of concept level, we decided to focus on the security aspect.

Additionally, Figure 37 shows that for larger rulesets, the interaction with the trust agent and the ruleset runtime represent a higher percentage of the time spent evaluating platform trust. As the amount of rules increases, so does the computing complexity and the communication with the attestation server. As each rule in the ruleset is run, events are sent to the attestation server for recording a log of the intermediate results, which increases the amount of time spent sending requests to the attestation server. One way to improve the performance, would be to only report the final result to the attestation server. However, by doing so, we lose the ability to inspect intermediate results upon an attestation failure, which makes the attestation forensics process difficult.

6.2 Remote attestation and whitelisting systems

A large amount of research has gone into identifying the challenges and threats in NFV [17, 44, 38, 35, 11]. Most of the attacks on NFV stem from the virtualization vulnerabilities and threats to network functions. Many of them will use known vulnerabilities as the infection vector for attacking and tampering with the system.

One approach to prevent attacks due to vulnerabilities in software (e.g. outdated hypervisors) is to use a whitelisting system, such as a package manager, which contains a list of packages/software signed by a trusted third party, which is considered safe to use. One may think that having such system in place would render measured boot and remote attestation useless. However, there are two reasons why this is not the case. First, measured boot allows us to reason over the status of the hardware of the platform, which we cannot do with the code repository, since it only controls software that runs on top of the OS, so anything before the OS in the boot chain is potentially harmful. Also, whitelisting systems may fail. These package repositories (e.g. windows software center, app stores, etc.) are based on the use of signed cryptographic certificates of the code by a trusted third party. However, there are situations in which malware may appear to be legitimate, because it is signed with proper keys. There is a market for code signing keys which sell for X amount of money in the dark web [28, 30]. These would allow malicious attackers to sign code which contains malware and it would go undetected by the whitelisting systems.

Since whitelisting systems are not foolproof, we consider them a complementary technology to remote attestation and measured boot. These should be combined to provide guarantees that the integrity of the system is not compromised. For example, even if a whitelisting/patch management system is set in place, we can still use remote attestation and system measurements to verify that the correct code is being

updated/installed and not just an update that appears to be valid because it was signed by a trusted key (as explained in Chapter 3, Section 3.4).

Although most of the infection vectors for malicious code are known vulnerabilities, there is still the possibility that a malicious attacker exploits a zero-day vulnerability to attack the system (e.g. install a rootkit or APT). In this situation, remote attestation may be useful to detect changes in the system and investigate the problems. Also, we have found research that combines trusted computing technologies and APT detection to trick the attacker that installs APT to carry out their attack while notifying the administrator so that mitigations can be set in place or even investigate how the attack is carried out after the infiltration stage [6].

6.3 Summary

In this chapter, we evaluated the work in this thesis with regards to both the identified limitations of existing research and performance. For the performance evaluation, we discussed alternatives to increase the performance of the system. Finally, we analyzed how remote attestation and whitelisting systems are related, as well as how they can be combined to improve the security of our system.

7 Conclusions and Future Work

This chapter concludes the thesis and outlines different areas this work can extend to in the future. We summarize the research aims and results of this thesis. Finally, we introduce different areas of research this work can be extended to.

7.1 Conclusions

This thesis has managed to identify weaknesses in existing remote attestation schemes and use them as motivation to design and implement a new remote attestation solution. The main motivation behind this work is to improve the concept of trust in NFV. This thesis focuses on trust in the underlying NFVI layer of NFV, since this is the building block for establishing trust at higher levels.

We extend the measurements we can obtain from a system to include run-time measurements, by providing a mechanism to measure files in the filesystem. This mechanism, combined with TPM, allows us to detect changes in critical files at runtime. Furthermore, we defined a mechanism to seal the configuration needed by our agents running on the monitored machines to platform state. Therefore, those agents can only run correctly if the platform is a correct state.

Additionally, we have designed and implemented an attestation server, which can store historic information about the devices it monitors. This historic information includes measurements, events and trust decisions.

To improve the existing definition of trust, we designed a rule system on our attestation server, which evaluates platform trust according to rulesets. These rulesets utilize the measurements from the devices, as well as events stored in the attestation server to determine if an element is trusted or not. Existing work only considers PCR values when evaluating trust. In contrast, the rules in our rule system reason over the PCR values, as well as the extra metadata included in TPM 2.0 quotes. Our rule system reasons over the quotes in a temporal manner, evaluating how the fields in the quotes have changed over time. Additionally, it takes into consideration the events that have happened in the system and how those may have affected the quotes. The introduction of our rule system allows us to have a finer-grained definition of trust. Additionally, the rules in our rule system are extensible, which gives the system the flexibility to adapt to new system requirements, as well as known misbehaviours.

Another benefit of our rule system is that the rule sets that compose it can be ordered according to strictness, which, effectively, produces a trust hierarchy. The trust hierarchy allows us to evaluate different machines according to different definitions of trust. We can use the stricter definitions of trust to determine what devices should run critical workload. By allowing for weaker definitions of trust, we can still use the

resources of machines that would be considered untrusted by other remote attestation schemes, e.g. by allowing them to run non-critical workload.

Finally, by means of root cause analysis, we identified a set of causal factor trees and Ishikawa diagrams that illustrate the common causes for failures in our platforms. These diagrams can be used as a tool in attestation forensics to investigate the causes of failures. Furthermore, the root causes of the failures can be mapped to mitigations to handle the erroneous state.

7.2 Future work

In this section, we outline different directions this work can take in the future.

This work focuses on trust at the NFVI layer of the NFV architecture. The concepts introduced in this work, such as the rule system, event system and RCA trees, can be extended to the VNF and MANO levels. The attestation server is designed in such a way that it would be possible for it to monitor elements from other layers of the NFV architecture. The rule system can be extended with new rules that define trust for those different levels of the architecture. This would extend the finer-grained definition of trust to other components of NFV.

In this thesis, we discussed how the attestation server can integrate with external systems. The attestation server can be extended by creating a trust graph, with the information of which elements trust each other. The information in this trust graph can be utilized by external mechanisms. For example, if two elements do not match the required level of trust, then, we can introduce fallback mechanisms, such as altering the routing network graph to provide safer routing [34, 33].

The RCA tools presented in this work are at an early stage. Future work includes automating the analysis of attestation failures. We can create an automated system that takes a trust failure, evaluates what the possible root cause is according to known rules (the defined trees) and triggers mitigations accordingly.

Finally, the remote attestation process can be distributed to all the machines in the system. By using longer term identities [5] for the elements and distributed ledger technology (e.g. Blockchain) to store attestation information, we can distribute the functionality of the attestation server to the elements. Therefore, each element can request any other element to quote itself and the trust status of an element can be requested from the distributed ledger.

References

[1] "CVE-2017-15361," Oct. 2017. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-15361

[2] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-FLAT: Control-FLow ATtestation for Embedded Systems Software," May 2016. [Online]. Available: http://arxiv.org/abs/1605.07763

[3] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A.-R. Sadeghi, and M. Schunter, "SANA: Secure and Scalable Aggregate Network Attestation," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*. New York, New York, USA: ACM Press, 2016, pp. 731–742. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2976749.2978335

[4] ARM Limited, "ARM Security Technology: Building a Secure System using TrustZone Technology," Whitepaper, 2009. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[5] D. Augot, H. Chabanne, Thomas Chenevier, W. George, and L. Lamber, "A User-Centric System for Verified Identities on the Bitcoin Blockchain," *arXiv:1710.02019 [cs, math]*, Oct. 2017. [Online]. Available: http://arxiv.org/abs/1710.02019

[6] R. P. Baksi and S. J. Upadhyaya, "Kidemonas: The Silent Guardian," vol. abs/1712.00841, p. 6, 2017. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1712/1712.00841.pdf

[7] M. D. Benedictis and A. Lioy, "On the establishment of trust in the cloud-based ETSI NFV framework," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2017, pp. 280–285.

[8] P. Bosch, A. Duminuco, F. Pianese, and T. L. Wood, "Telco clouds and Virtual Telco: Consolidation, convergence, and beyond," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, May 2011, pp. 982–988.

[9] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "TyTAN: Tiny Trust Anchor for Tiny Devices," in *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*. New York, New York, USA: ACM Press, 2015, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2744769.2744922

[10] N. Cardoso and R. Abreu, "Self-Healing on the Cloud: State-of-the-Art and

Future Challenges," in *2012 Eighth International Conference on the Quality of Information and Communications Technology*, Sep. 2012, pp. 279–284.

[11] M. Daghmehchi Firoozjaei, J. P. Jeong, H. Ko, and H. Kim, "Security challenges with network functions virtualization," *Future Generation Computer Systems*, vol. 67, pp. 315–324, Feb. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16302321

[12] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A.-R. Sadeghi, "LO-FAT: Low-Overhead control Flow ATtestation in hardware," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Jun. 2017, pp. 1–6.

[13] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "The Untapped Potential of Trusted Execution Environments on Mobile Devices," *IEEE Security Privacy*, vol. 12, no. 4, pp. 29–37, Jul. 2014.

[14] ETSI, "Network Functions Virtualization - Introductory White Paper," Tech Report, 2012. [Online]. Available: https://portal.etsi.org/nfv/nfv_white_paper.pdf

[15] ETSI ISG NFV, "Network Functions Virtualisation (NFV); Architectural Framework," Group Specification ETSI GS NFV 002 V1.2.1, Dec. 2014. [Online]. Available: https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV%20002v1.2.1%20-%20GS%20-%20NFV%20Architectural%20Framework.pdf

[16] ETSI ISG NFV-REL, "Network Functions Virtualisation (NFV); Resiliency Requirements," Group Specification ETSI GS NFV-REL 001 V1.1.1, Jan. 2015. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.01_60/gs_NFV-REL001v010101p.pdf

[17] ETSI ISG NFV-SEC, "Network Functions Virtualisation (NFV); NFV Security; Problem Statement," Group Specification ETSI GS NFV-SEC 001 V1.1.1, Oct. 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/001/01.01.01_60/gs_NFV-SEC001v010101p.pdf

[18] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "A Minimalist Approach to Remote Attestation," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. Proceedings, Design, Automation & Test in Europe. Dresden, Germany: European Design and Automation Association, Mar. 2014, pp. 244:1–244:6. [Online]. Available: https://dl.acm.org/citation.cfm?id=2616905&CFID=996740229&CFTOKEN=49778175&qualifier=LU1041897

[19] D. Fu and X. Peng, "TPM-based remote attestation for Wireless Sensor Networks," *Tsinghua Science and Technology*, vol. 21, no. 3, pp. 312–321, Jun.

2016.

[20] J. Greene, "Intel® Trusted Execution Technology: White Paper," Tech. Rep., 2012. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html

[21] Intel, "Open CIT 3.2.1 Product Guide." [Online]. Available: https://github.com/opencit/opencit/wiki/Open-CIT-3.2.1-Product-Guide

[22] Intel Corporation, "Open Cloud Integrity Technology (Open CIT)." [Online]. Available: https://01.org/opencit

[23] Intel Corporation, "Trusted Boot (tboot)." [Online]. Available: https://sourceforge.net/projects/tboot/

[24] Intel Corporation, "OpenAttestation," Sep. 2013. [Online]. Available: https://01.org/openattestation

[25] International Electrotechnical Commission (IEC), "IEC 31025: Fault Tree Analysis (FTA)," International Standard, 2006.

[26] K. Ishikawa, *Guide to Quality Control*, ser. Industrial engineering and technology. Asian Productivity Organization, 1986. [Online]. Available: https://books.google.fi/books?id=POEeAQAAIAAJ

[27] L. Jacquin, A. Lioy, D. R. Lopez, A. L. Shaw, and T. Su, "The Trust Problem in Modern Network Infrastructures," in *Cyber Security and Privacy*. Springer, Cham, Apr. 2015, pp. 116–127. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-25360-2_10

[28] D. Kim, B. J. Kwon, and T. Dumitraş, "Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI." ACM Press, 2017, pp. 1435–1448. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3133956.3133958

[29] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "TrustLite: a security architecture for tiny embedded devices." ACM Press, 2014, pp. 1–14. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2592798.2592824

[30] K. Kozák, B. J. Kwon, D. Kim, C. Gates, and T. Dumitraş, "Issued for Abuse: Measuring the Underground Trade in Code Signing Certificate," *arXiv:1803.02931 [cs]*, Mar. 2018. [Online]. Available: http://arxiv.org/abs/1803.02931

[31] M. Kylänpää and A. Rantala, "Remote Attestation for Embedded Systems," *Security of Industrial Control Systems and Cyber Physical Systems*, pp. 79–92, Sep. 2015. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-40385-4_6

[32] P. B. Ladkin, "Causal Reasoning about Aircraft Accidents," in *Computer Safety, Reliability and Security*, ser. Lecture Notes in Computer Science, F. Koornneef and M. van der Meulen, Eds. Springer Berlin Heidelberg, 2000, pp. 344–360.

[33] S. Lal, A. Kalliola, I. Oliver, K. Ahola, and T. Taleb, "Securing VNF communication in NFVI," in *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*, Sep. 2017, pp. 187–192.

[34] S. Lal, S. Ravidas, I. Oliver, and T. Taleb, "Assuring virtual network function image integrity and host sealing in Telco cloue," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[35] S. Lal, T. Taleb, and A. Dutta, "NFV: Security Threats and Best Practices," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 211–217, 2017.

[36] A. Lioy, T. Su, A. L. Shaw, H. Attak, D. R. Lopez, and A. Pastor, "Trust in SDN/NFV Environments," in *Guide to Security in SDN and NFV*, S. Y. Zhu, S. Scott-Hayward, L. Jacquin, and R. Hill, Eds. Cham: Springer International Publishing, 2017, pp. 103–124. [Online]. Available: http://link.springer.com/10.1007/978-3-319-64653-4_4

[37] K. N. McGill, "Trusted Mobile Devices: Requirements for a Mobile Trusted Platform Module," *JOHNS HOPKINS APL TECHNICAL DIGEST*, vol. 32, no. 2, p. 11, 2013.

[38] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7243304/

[39] I. Oliver, A. Kalliola, S. Holtmanns, Y. Miche, G. Limonta, B. Vigmostad, and K. Muller, "A Testbed for Trusted Telecommunications Systems in a Safety Critical Environment," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science, B. Gallina, A. Skavhaug, E. Schoitsch, and F. Bitsch, Eds. Springer International Publishing, 2018, pp. 87–98.

[40] I. Oliver, G. Limonta, and B. Vigmostad, "Improving System Trustworthiness by Combining Remote Attestation and Root Cause Analysis," in *11th International Conference on the Quality of Information and Communications Technology - QUATIC 2018*, Coimbra, Portugal, Sep. 2018.

[41] J. D. Osborn and D. C. Challener, "Trusted Platform Module Evolution," *JOHNS HOPKINS APL TECHNICAL DIGEST*, vol. 32, no. 2, p. 8, 2013.

[42] S. Ravidas, "Incorporating Trust in Network Function Virtualization," Master's thesis, Aalto University, Espoo, Oct. 2016.

[43] S. Ravidas, S. Lal, I. Oliver, and L. Hippelainen, "Incorporating trust in NFV: Addressing the challenges," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, Mar. 2017, pp. 87–91.

[44] F. Reynaud, F.-X. Aguessy, O. Bettan, M. Bouet, and V. Conan, "Attacks against Network Functions Virtualization and Software-Defined Networking: State-of-the-art," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, Jun. 2016, pp. 471–476.

[45] D. Safford and M. Zohar, "Extending the Linux Integrity Subsystem for TCB Protection," 2014. [Online]. Available: http://kernsec.org/files/lss2014/safford_tcb_integrity.pdf

[46] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture," *13th USENIX Security Symposium*, p. 17, 2004.

[47] S. Schulz, A. Schaller, F. Kohnhäuser, and S. Katzenbeisser, "Boot Attestation: Secure Remote Reporting with Off-The-Shelf IoT Sensors," Tech. Rep. 577, 2017. [Online]. Available: http://eprint.iacr.org/2017/577

[48] SELinux Wiki, "Main Page — SELinux Wiki," 2017. [Online]. Available: https://selinuxproject.org/page/Main_Page

[49] TCG, "Trusted Platform Module Library, Part 1: Architecture," The Trusted Computing Group, Trusted Platform Module Library Specification, Family 2.0 Level 00, Revision 01.38, Sep. 2016. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf

[50] TCG, "Trusted Platform Module Library, Part 2: Structures," The Trusted Computing Group, Trusted Platform Module Library Specification, Family 2.0 Level 00, Revision 01.38, Sep. 2016. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-2-Structures-01.38.pdf

[51] TCG, "TSS TAB and Resource Manager Specification," Trusted Computing Group, TSS TAB and Resource Manager Specification, Family 2.0 Level 00, Revision 01.00, Mar. 2016. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TSS-2.0-TAB-Resource-Manager-SpecVer1.0-Rev18_review_END030918.pdf

[52] TCG, "TCG PC Client Platform Firmware Profile," TCG PC Client Platform Firmware Profile Specification, Family 2.0 Level 00 Revision 1.03 v51, May 2017. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/PC-ClientSpecific_Platform_Profile_for_TPM_2p0_Systems_v51.pdf

[53] TCG, "TSS 2.0 Overview and Common Structures," The Trusted

Computing Group, TCG TSS 2.0 Overview and Common Structures Specification Version 0.90, Revision 03, Jan. 2018. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCG-TSS-2.0-Overview-and-Common-Structures-Specification-Version-0.90-Revision-02.pdf

[54] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, "Fault Tree Handbook (NUREG-0492)," p. 209, 1981. [Online]. Available: https://www.nrc.gov/docs/ML1007/ML100780465.pdf

[55] B. O. Vigmostad, "Enhancing Trust and Resource Allocation in Telecommunications Cloud," Master's thesis, Aalto University, Espoo, Finland, Sep. 2018.

[56] R. Wilkins and B. Richardson, "UEFI Secure Boot in Modern Computer Security Solutions," p. 10, Sep. 2013. [Online]. Available: https://www.uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf

[57] G. Xu, Y. Tang, Z. Yan, and P. Zhang, "TIM: A Trust Insurance Mechanism for Network Function Virtualization Based on Trusted Computing," in *Security, Privacy, and Anonymity in Computation, Communication, and Storage*. Springer, Cham, Dec. 2017, pp. 139–152. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-72389-1_13

[58] T. Zhang and R. B. Lee, "CloudMonatt: an Architecture for Security Health Monitoring and Attestation of Virtual Machines in Cloud Computing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture - ISCA '15*, vol. 43. New York, New York, USA: ACM Press, 2015, pp. 362–374. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2749469.2750422

[59] M. Zohar, "IMA policies," May 2008. [Online]. Available: https://www.kernel.org/doc/Documentation/ABI/testing/ima_policy

A Static Rules

In our rule system, we define six static rules.

A.1 Correct attested value

This rule compares the 'attested' field of the quote to the expected value stored in the attestation database for the policy this quote is for. The 'attested' field contains a digest of the contents of the given PCRs. The policy for this quote stores the pcrs and the expected 'attested' value for that PCR combination.

$$\text{correctAttestedValue}(q, p) = \begin{cases} \text{True}, & q.attested = p.expectedValue \\ \text{False}, & \text{otherwise} \end{cases} \tag{A1}$$

Given an element e, a quote q and a policy p for q, and assuming that e follows p, we define the logic of the rule in Equation A1. The rule will be satisfied when the attested value matches the expected value, and it will fail when it does not match.

A.2 Valid signature

This rule checks the validity of the 'signature' field of a quote. Every quote includes a signature value, which is a signature over the returned quote structure by the attestation key of the element [49]. The attestation database stores the public attestation key for all elements in the system.

$$\text{validSignature}(e, q, s) = \begin{cases} \text{True}, & \text{verifySignature}(q, s, e.ak) \\ \text{False}, & \text{otherwise} \end{cases} \tag{A2}$$

Therefore, given an element e, a quote q for e and a signature s for q, we define the logic of the rule in Equation A2. The rule will be satisfied if s is a valid signature for q using $e.ak$, it fails when the signature cannot be verified. The attestation key is unique for every TPM. If a quote for an element is not signed by the attestation key registered to the element, we cannot guarantee that the element's TPM generated this quote, which would make the element untrusted.

A.3 Valid safe value

When an element is quoted, the quote includes a 'clock' and a 'safe' field. The clock of the TPM should increase monotonically. However, there are cases in which the clock may appear to go backwards. This can happen when there is a power outage or some other kind of non-orderly shutdown. The safe value in the quote indicates whether the reported clock value is a new value or a repeat of any previously reported value [49]. Safe is set to 1 when the clock integrity is reliable and 0 otherwise.

$$\text{validSafeValue}(q) = \begin{cases} \text{True}, & q.safe = 1 \\ \text{False}, & \text{otherwise} \end{cases} \tag{A3}$$

Given an element e and a quote q for e, we define the logic of the rule in Equation A3. This rule will be satisfied when the reported safe value for a quote is reliable.

A.4 Invalid safe value

This rule is defined to handle the situation in which the clock appears to go backwards, and the TPM has set the safe value to 0 to indicate that the clock value may be unreliable. This rule is meant to be used in conjunction with a temporal rule that checks whether the clock is increasing or not.

$$\text{invalidSafeValue}(q) = \begin{cases} \text{True}, & q.safe = 0 \\ \text{False}, & \text{otherwise} \end{cases} \tag{A4}$$

Given an element e and a quote q for e, we define the logic of the rule in Equation A4. This rule will be satisfied when the reported safe value for a quote is unreliable.

A.5 Valid type value

The 'type' field of a quote indicates that this is an attestation structure. For a TPM2_Quote, the type value is defined as `0x8018` [50]. If the type value is anything other than `0x8018`, this may be an indicator that the structure generated by the TPM is not a quote or that the quote was somehow forged.

$$\text{validTypeValue}(q) = \begin{cases} \text{True}, & q.type = \texttt{0x8018} \\ \text{False}, & \text{otherwise} \end{cases} \tag{A5}$$

Given an element e and a quote q for e, we define the logic of the rule in Equation A5. This rule will be satisfied when the quote contains the correct type value, and it will fail otherwise.

A.6 Valid magic value

The 'magic' value of a quote indicates that the structure was generated by a TPM. This value is always `0xFF'TCG'` [50]. If the magic value is anything other than the aforementioned value, this may be an indicator that the quote was not generated by a TPM.

$$\text{validMagicValue}(q) = \begin{cases} \text{True}, & q.magic = \texttt{0xFF'TCG'} \\ \text{False}, & \text{otherwise} \end{cases} \tag{A6}$$

Given an element e and a quote q for e, we define the logic of the rule in Equation A6. This rule will be satisfied when the quote contains the correct magic value, and it will fail otherwise.

A.7 Valid firmware

The 'firmware' value of a quote indicates the firmware version for that element's TPM. We have a set of known firmware values that the TPMs can have. If the firmware value is not one of the known values for the TPMs in our elements, this may indicate some kind of tampering.

$$\text{validFirmware}(q, F) = \begin{cases} \text{True}, & q.firmware \in F \\ \text{False}, & \text{otherwise} \end{cases} \tag{A7}$$

Given a quote e and a set of known firmware values F, we define the logic of the rule in Equation A7. This rule will be satisfied when the firmware value for the TPM reported by the quote is in the set of known values, and it will fail otherwise.

B Temporal Rules

In our rule system, we define twenty-one temporal rules.

B.1 Magic value not changed

This rule checks that the magic value has not changed from one quote to another.

$$\text{magicNotChanged}(q, q') = \begin{cases} \text{True}, & q.magic = q'.magic \\ \text{False}, & \text{otherwise} \end{cases} \tag{B1}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B1. This rule will be satisfied when the magic value has not changed between two quotes, and it will fail if it does change.

B.2 Type value not changed

This rule checks that the type value has not changed from one quote to another.

$$\text{typeNotChanged}(q, q') = \begin{cases} \text{True}, & q.type = q'.type \\ \text{False}, & \text{otherwise} \end{cases} \tag{B2}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B2. This rule will be satisfied when the type value has not changed between two quotes, and it will fail if it does change.

B.3 Firmware version not changed

In the static rules, we check that the firmware version reported by a quote is one of the known firmware versions for our TPMs. The firmware version of a TPM should not change from one quote to another without a good reason. If the firmware version changed, this means someone has updated the firmware on the machine, which is not considered normal behaviour. This rule checks that the firmware version is the same between two quotes.

$$\text{firmwareNotChanged}(q, q') = \begin{cases} \text{True}, & q.firmware = q'.firmware \\ \text{False}, & \text{otherwise} \end{cases} \tag{B3}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B3. This rule will be satisfied when the firmware value has not changed between two quotes, and it will fail if it does change.

B.4 Qualified signer not changed

A quote includes a field called 'qualifiedSigner', which indicates the name of the key used to sign the quote [49]. In our system, the Attestation Key of the TPM is used to sign the quote. Since we do not ask the TPM to sign attestation data with any other key, we expect the qualified signer to be the same for every quote.

$$\text{qualifiedSignerNotChanged}(q, q') = \begin{cases} \text{True}, & q.qualifiedSigner = q'.qualifiedSigner \\ \text{False}, & \text{otherwise} \end{cases} \tag{B4}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B4. This rule will be satisfied when the qualified signer value has not changed between two quotes, and it will fail if it does change.

B.5 Clock Rules

TPMs keep a time value in volatile memory and periodically stores a copy of it on NV memory [49]. Quotes report this time value on their 'clock' field. The clock value is monotonically increasing. If a quote reports a clock value that is lower than that of a previous quote, this may indicate that the element has suffered a non-orderly shutdown.

We define two rules that can be used to evaluate the status of the TPM clock. In most cases, the clock is expected to increase.

$$\text{clockIncreased}(q, q') = \begin{cases} \text{True}, & q.clock < q'.clock \\ \text{False}, & \text{otherwise} \end{cases} \tag{B5}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B5. This rule will be satisfied when the clock has increased between two quotes, and it will fail if it stays the same or decreases.

We define a second rule over the clock, which checks if the clock has decreased. It can be used in conjunction with other rules to detect situations in which elements

are not properly shut down.

$$\text{clockDecreased}(q, q') = \begin{cases} \text{True}, & q.clock > q'.clock \\ \text{False}, & \text{otherwise} \end{cases} \tag{B6}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B6. This rule will be satisfied when the clock has decreased between two quotes, and it will fail if it stays the same or increases.

B.6 Signature changed

Every time an element is quoted, it will report at least a different clock value. Therefore, the signature for the quote will change from quote to quote. This rule allows us to detect if any signature is being reused between quotes.

$$\text{signatureChanged}(q, q') = \begin{cases} \text{True}, & q.signature \neq q'.signature \\ \text{False}, & \text{otherwise} \end{cases} \tag{B7}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B7. This rule will be satisfied when the signatures are not the same, and it will fail if they are.

B.7 Reset count rules

The 'resetCount' field of a quote is a counter that increases every time an element is rebooted [49]. It is reset to zero when the TPM is cleared. During the normal lifecycle of an element, the reset count should either stay the same or increase. We define two rules regarding the reset count, one for each possible situation: the reset count increasing or not changing. If the reset count decreases, it may be an indicator that the TPM was reset to the factory defaults.

B.7.1 Reset count has not changed

This rule checks that the number of reboots of the element has not changed. This rule can be used to check that an element is operating in a normal way and has not

been rebooted.

$$\text{resetCountNotChanged}(q, q') = \begin{cases} \text{True}, & q.resetCount = q'.resetCount \\ \text{False}, & \text{otherwise} \end{cases} \tag{B8}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B8. This rule will be satisfied when the reset count has stayed the same between two quotes, and it will fail if it changes.

B.7.2 Reset count has increased

This rule checks that the number of reboots of the element has increased. This rule is used to verify that an element has been rebooted.

$$\text{resetCountIncreased}(q, q') = \begin{cases} \text{True}, & q.resetCount < q'.resetCount \\ \text{False}, & \text{otherwise} \end{cases} \tag{B9}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B9. This rule will be satisfied when the reset count has increased between two quotes, and it will fail if it decreases or stays the same.

B.7.3 Reset count matches reboot events

Under normal operation, the amount of reboot events should match the reset count difference reported by the TPM between quotes. We define a rule that can verify this situation.

$$\text{resetCountMatchesReboots}(q, q', B) = \begin{cases} \text{True} & (q'.resetCount - q.resetCount) = |B| \\ \text{False} & \text{otherwise} \end{cases} \tag{B10}$$

Given an element e and a pair of quotes for e, q and q', assuming q was taken before q', we define the logic of the rule in equation B10 This rule will be satisfied when the amount of reboots reported by an element matches the reboot events between quotes and fail otherwise.

B.7.4 Reset count higher reboot events

In some situations, the TPM will report a higher amount of reboots than the ones stored in the attestation database. This may be a valid situation. For example, portable devices, such as laptops, usually connect to networks outside the corporate one. If a machine is rebooted outside the network, it may fail to communicate this event to the attestation server. Even though the attestation server will not know a reboot happened, the reboot will still be reflected in the TPM reset counter. Since this is a known situation for a specific set of devices, we can define a rule that is used with these devices, which would allow the reset count to be higher than the amount of reboots.

$$\text{resetCountHigherThanReboots}(q, q', B) = \begin{cases} \text{True} & (q'.resetCount - q.resetCount) > |B| \\ \text{False} & \text{otherwise} \end{cases} \tag{B11}$$

Given an element e and a pair of quotes for e, q and q', assuming q was taken before q', we define the logic of the rule in equation B11 This rule will be satisfied when the amount of reboots reported by an element is higher than the reboot events between quotes and fail otherwise.

B.8 Restart count rules

The 'restartCount' field of a quote is a counter that increases when an element is suspended or hibernated. The rules defined in this section are used to check if the element has been suspended or hibernated.

When a TPM is started up back from hibernation, it resets all the PCRs to their initial values and measures the boot sequence once again [49]. Therefore, it is possible to do changes on the startup sequence, such as booting to a different kernel version or changing BIOS settings. This field helps us detect these situations.

Additionally, some TPM chips present failures when waking up from a suspension or hibernation state. After waking up, the TPM can no longer be used until a reboot happens. In these kind of elements, the restart count stays the same until the next boot, since once the device goes to sleep, it is not possible to quote it anymore before restarting the system.

B.8.1 Restart count not changed

This rule checks the case in which an element has not been suspended or hibernated.

$$\text{restartCountNotChanged}(q, q') = \begin{cases} \text{True}, & q.restartCount = q'.restartCount \\ \text{False}, & \text{otherwise} \end{cases} \tag{B12}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B12. This rule will be satisfied when the restart count has not changed between two quotes, and it will fail if it does change.

B.8.2 Restart count increased

This rule is used to verify that the element has been suspended or hibernated. On machines with TPM chips that work correctly, it is acceptable behaviour to suspend or hibernate (e.g. laptops).

$$\text{restartCountIncreased}(q, q') = \begin{cases} \text{True}, & q.restartCount < q'.restartCount \\ \text{False}, & \text{otherwise} \end{cases} \tag{B13}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B13. This rule will be satisfied when the restart count has increased between two quotes, and it will fail if it stays the same or decreases.

B.8.3 Restart count decreased

This rule is used when the restart count decreases between quotes. This may happen when a machine, which had previously been suspended or hibernated, is rebooted. After the reboot, the reset count increases and the restart count is reset, which would appear to be a decrease in the counter. This rule can be used with rule B9 to detect this situation.

$$\text{restartCountDecreased}(q, q') = \begin{cases} \text{True}, & q.restartCount > q'.restartCount \\ \text{False}, & \text{otherwise} \end{cases} \tag{B14}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B14. This rule will be satisfied when the restart count has decreased between two quotes, and it will fail if it stays the same or decreases.

B.9 Attested value not changed

This rule detects if the attested value has stayed the same between two quotes. For a trusted element, we would expect for all its quotes to have a correct attested value (rule A1), and this attested value should not change. This rule allows us to verify the correct behaviour of the system.

$$\text{attestedNotChanged}(q, q') = \begin{cases} \text{True}, & q.attested = q'.attested \\ \text{False}, & \text{otherwise} \end{cases} \tag{B15}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B15. This rule will be satisfied when the attested value has not changed between two quotes, and it will fail if it does.

B.10 Attested value changed

This rule detects if the attested value has changed between two quotes. Normally, we would expect the attested value to be correct and not change. However, if the software on that element is updated, the attested value may change. This rule allows us to detect this situation.

$$\text{attestedChanged}(q, q') = \begin{cases} \text{True}, & q.attested \neq q'.attested \\ \text{False}, & \text{otherwise} \end{cases} \tag{B16}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B16. This rule will be satisfied when the attested value has changed between two quotes, and it will fail if it does not.

B.11 Quote changed

Every quote structure for an element should be different. The quote contains fields that change every time the element is quoted, such as the clock and the signature. This rule allows us to detect if any quote is being reported more than once. Therefore,

a malicious actor cannot obtain one good quote and report that back to the attestation server whenever the element is quoted.

$$\text{quoteChanged}(q, q') = \begin{cases} \text{True}, & q \neq q' \\ \text{False}, & \text{otherwise} \end{cases} \tag{B17}$$

Given a pair of quotes q and q' and assuming q was taken before q', we define the logic of the rule in Equation B17. This rule will be satisfied when the quotes are not the same, and it will fail if they are.

B.12 Policy changed

Every time the expected value of a policy is changed, we store an 'update' event in the attestation database. This rule reasons over the 'update policy' events that may have happened between two quotes. It allows us to detect situations in which a policy for an element has changed its expected value in the time interval between two quotes. Combined with other rules, it allows us to reason over the changes we encounter between quotes.

$$\text{policyChanged}(p, q, q') = \begin{cases} \text{True}, & (\exists\ u \in \mathit{getEvents}(q.\mathit{timestamp}, q'.\mathit{timestamp}) \mid \\ & \quad u.\mathit{type} = \text{'policy update'} \wedge u.\mathit{policy} = p) \\ \text{False}, & \text{otherwise} \end{cases}$$

where:

$$\mathit{getEvents}(t, t') = \{e \mid t \leq e.\mathit{timestamp} \leq t' \wedge e \text{ is an event in the attestation database}\} \tag{B18}$$

Given a policy p and pair of quotes for p, q and q', assuming q was taken before q', we define the logic of the rule in Equation B18. This rule will be satisfied when the policy has been updated between quotes, and it will fail otherwise.

B.13 Policy not changed

This rule is the opposite of rule B18. It is used to verify that a policy has not been updated between two quotes. Combined with other rules, it allows us to reason over the changes we encounter between quotes.

$$\text{policyNotChanged}(p, q, q') = \begin{cases} \text{True}, & \neg\ (\exists\ u \in getEvents(q.timestamp, q'.timestamp)\ | \\ & \quad u.type = \text{`policy update'} \wedge u.policy = p) \\ \text{False}, & \text{otherwise} \end{cases}$$

where:

$$getEvents(t, t') = \{e \mid t \leq e.timestamp \leq t' \wedge e \text{ is an event in the attestation database}\} \tag{B19}$$

Given a policy p and pair of quotes for p, q and q', assuming q was taken before q', we define the logic of the rule in Equation B19. This rule will be satisfied when the policy has not been updated between quotes, and it will fail otherwise.

B.14 Element updated

Every time an element is updated, we store an 'update' event in the attestation database. Similar to the previous rule, this rule reasons over 'update' events for an element that may have happened between two quotes. Combined with other rules, it allows us to detect situations in which changes in a quote are due to a software update on the element.

$$\text{elementUpdated}(e, q, q') = \begin{cases} \text{True}, & (\exists\ u \in getEvents(q.timestamp, q'.timestamp)\ | \\ & \quad u.type = \text{`element update'} \wedge u.element = e) \\ \text{False}, & \text{otherwise} \end{cases}$$

where:

$$getEvents(t, t') = \{e \mid t \leq e.timestamp \leq t' \wedge e \text{ is an event in the attestation database}\} \tag{B20}$$

Given an element e and pair of quotes for e, q and q', assuming q was taken before q', we define the logic of the rule in Equation B20. This rule will be satisfied when the element has been updated between quotes, and it will fail otherwise.

B.15 Element not updated

This rule is the opposite of rule B20. It is used to verify that an element has not been updated between two quotes. Combined with other rules, it allows us to reason over the changes we encounter between quotes.

$$\text{elementNotUpdated}(e, q, q') = \begin{cases} \text{True}, & \neg\ (\exists\ u \in getEvents(q.timestamp, q'.timestamp)\ | \\ & \quad u.type = \text{`element update'} \wedge u.element = e) \\ \text{False}, & \text{otherwise} \end{cases}$$

where:

$$getEvents(t, t') = \{e \mid t \leq e.timestamp \leq t' \wedge e \text{ is an event in the attestation database}\} \tag{B21}$$

Given an element e and pair of quotes for e, q and q', assuming q was taken before q', we define the logic of the rule in Equation B21. This rule will be satisfied when the element has not been updated between quotes, and it will fail otherwise.

C Compound Rules

In our rule system, we define three compound rules. For simplicity, we define compound rules as trees instead of equations, since compound rules use the results of their children rules to determine their own. There are two kinds of compound rules: AND and OR rules. The result of an AND rule is the logical AND operation over the results of the children rules. Similarly, the result of an OR rule is the logical OR operation over the result of the children rules. Each of the following rules indicates if it is an AND or OR rule.

C.1 Clock integrity not maintained (AND Rule)

This rule is used when the clock appears to be decreasing, but the TPM has marked the clock value as unreliable. Given a pair of quotes q and q' and assuming q was taken before q', Figure C1 shows the logic of the rule.

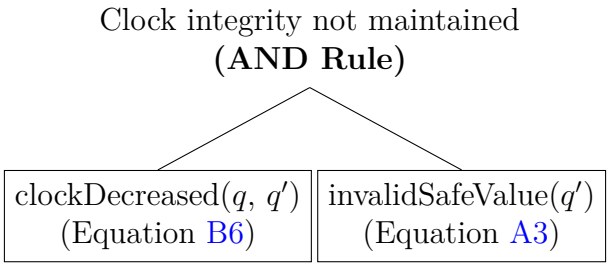

Figure C1: Clock integrity not maintained (AND rule)

C.2 Clock increasing or integrity not maintained (OR Rule)

This rule is used during normal operation of an element. There are two situations that may happen: the clock monotonically increases between quotes or the clock integrity is not maintained. This rule checks that one of these conditions holds. Given a pair of quotes q and q' and assuming q was taken before q', Figure C2 shows the logic of the rule.

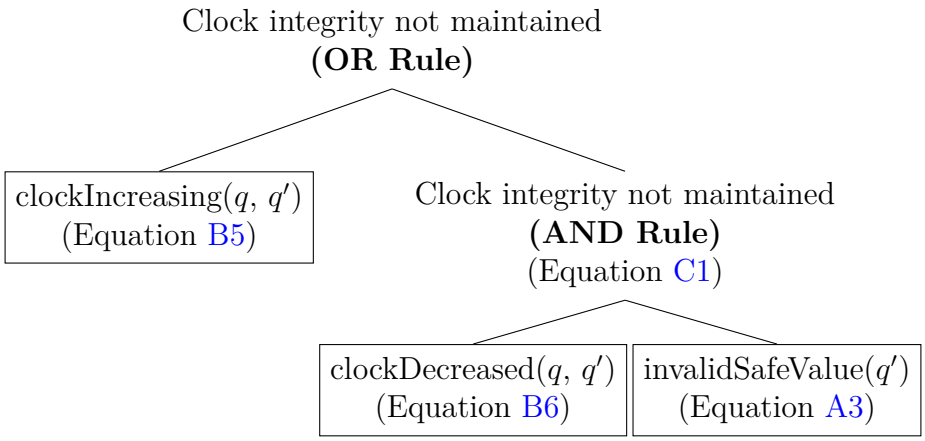

Figure C2: Clock increasing or integrity not maintained (OR rule)

C.3 Compound reboot rules

During the lifecycle of an element, we can identify three different situations that may happen between two quotes.

1. **Normal operation**: element has neither been rebooted nor suspended between quotes.
2. **Reboot:** element has been rebooted between quotes.
3. **Suspend or Hibernate:** element has been suspended or hibernated between quotes.

We define a set of compound rules that allow us to detect each of these situations. These will be used as building blocks for more detailed reasoning in other compound rules.

C.3.1 Element has not been rebooted (AND Rule)

During normal operation an element has neither been rebooted nor suspended; therefore, the reset and restart counters reported by the TPM should show no changes between quotes. Given a pair of quotes q and q' and assuming q was taken before q', Figure C3 shows the logic of the rule.

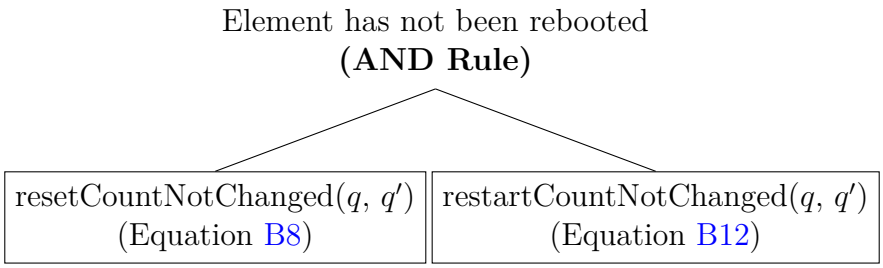

Figure C3: Element has not been rebooted (AND rule)

C.3.2 Element has been rebooted (AND Rule)

When an element is rebooted between quotes the reset counter should increase. The restart counter may stay the same or decrease. If the machine had been previously suspended, upon reboot, the restart counter will be reset to zero, which makes it seem as the counter is decreasing. If the machine had not been suspended, the restart counter will show no changes. Given a pair of quotes q and q' and assuming q was taken before q', Figure C4 shows the logic of the rule.

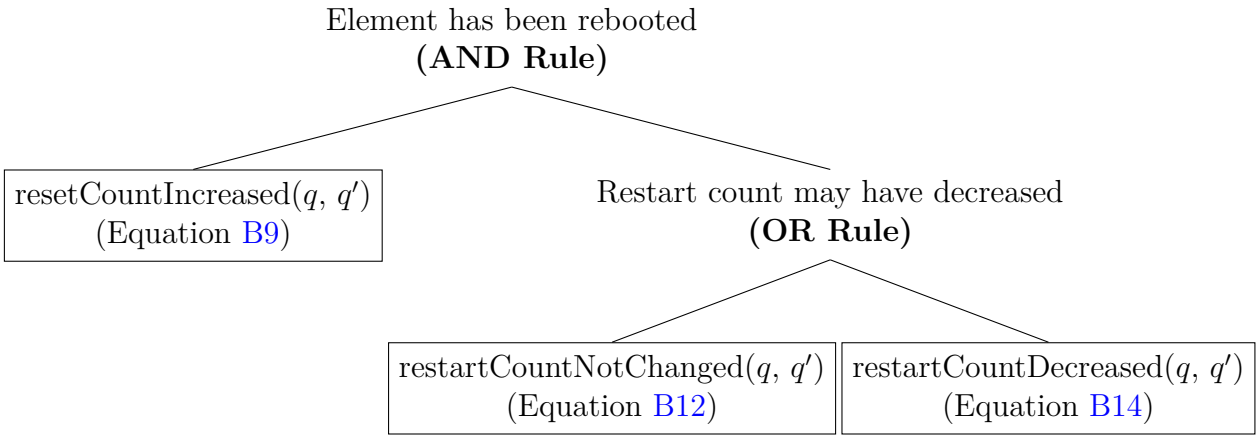

Figure C4: Element has been rebooted (AND rule)

C.3.3 Element has been suspended (AND Rule)

When an element is suspended between quotes, there will be no change in the reset count, and the restart count will increase by at least one. Given a pair of quotes q and q' and assuming q was taken before q', Figure C5 shows the logic of the rule.

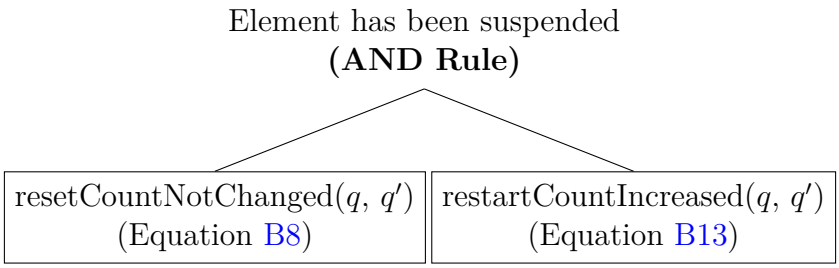

Figure C5: Element has been suspended (AND rule)

C.3.4 Reboot checks

We combine the previously defined rules by using an OR rule. Figure C6 shows the new compound rule, which checks that one of the previous situations has happened between quotes (normal operation, reboot or suspension).

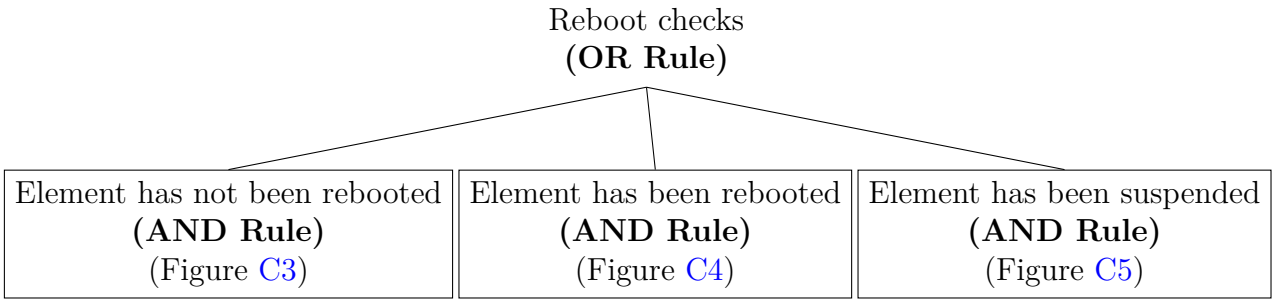

Figure C6: Reboot checks (OR rule)

C.4 Attested value checks

There are cases in which the attested value of an element is changed, due to software updates. We can define compound rules to handle these situations. We have identified five different situations that may arise when evaluating the events that occurred between quotes and their effect on the attested value reported by the TPM.

C.4.1 Normal operation

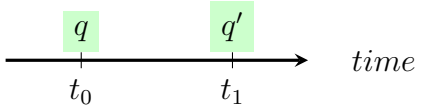

Figure C7: Normal operation timeline

We assume normal operation to be the situation in which no element updates or policy changes have happened between quotes as shown in Figure C7. In this case, the attested value should not change and it should satisfy the policy. Figure C8 shows the logic for the rule.

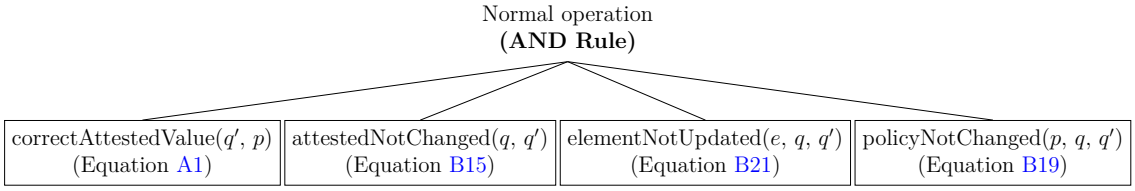

Figure C8: Normal operation (AND rule)

C.4.2 Element was updated and policy changed between quotes

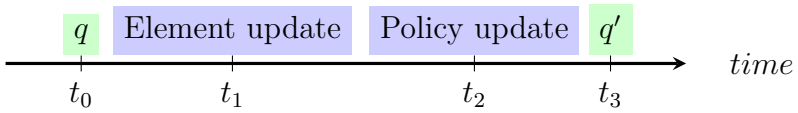

Figure C9: Element was updated and policy changed between quotes

Figure C9 shows the case when the element gets a software update, which affects one of its policies, and the policy is updated to reflect this change. Both updates happen between quotes q and q'. In this situation, the attested value changes; however, since the policy was updated, the latest quote will still satisfy the policy. Figure C10 shows the logic for the rule.

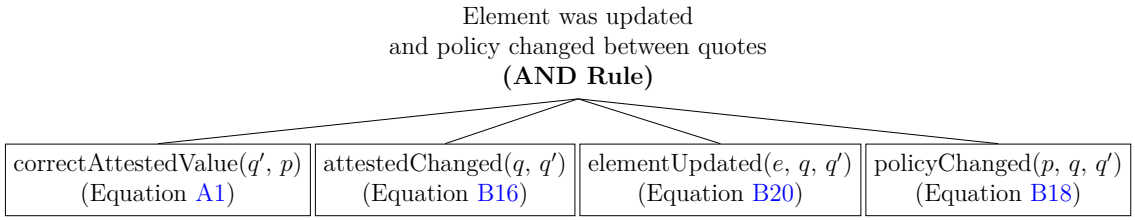

Figure C10: Element was updated and policy changed between quotes (AND rule)

C.4.3 Element was updated between quotes and policy had already changed

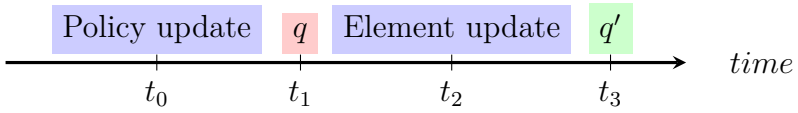

Figure C11: Element was updated between quotes and policy had already changed

Figure C11 shows the case when the policy of an element changes before the element is updated, which causes the first quote to not satisfy the policy anymore. Then, between q and q', the element is updated, which causes the quote to satisfy the policy again. We define this rule, to cover the case in which updates are done out of order, so the rule system knows this not a trust failure. Figure C12 shows the logic for the rule.

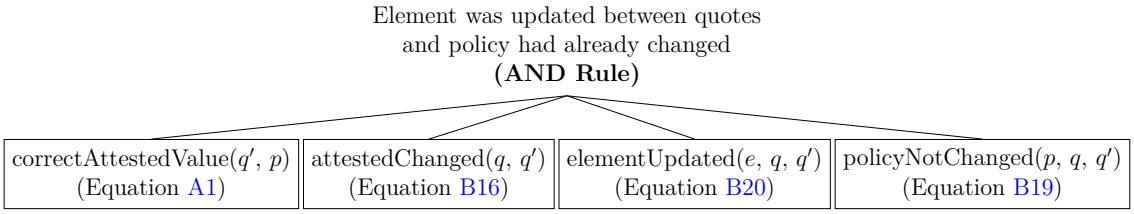

Figure C12: Element was updated between quotes and policy had already changed (AND rule)

C.4.4 Policy was changed between quotes and element had already been updated

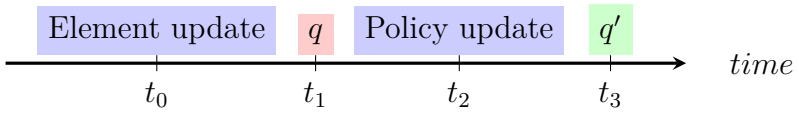

Figure C13: Element was updated between quotes and policy had already changed

Figure C13 shows the case when an element is updated before the policy is changed to reflect the update, which causes the first quote to not satisfy the policy anymore. Then, between q and q', the policy is updated, which causes the quote to satisfy the policy again. We define this rule, to cover the case in which updates are done out of order, so the rule system knows this not a trust failure. Figure C14 shows the logic for the rule.

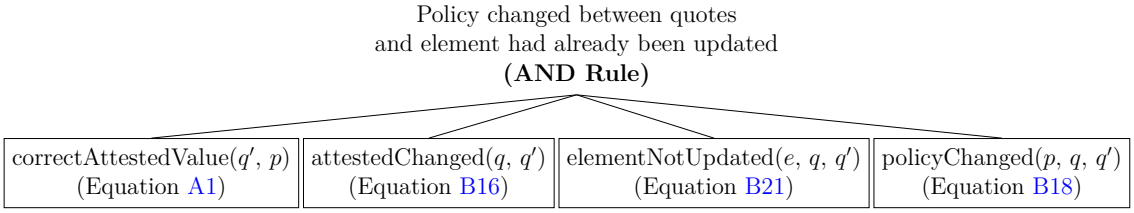

Figure C14: Policy changed between quotes and element had already been updated (AND rule)

C.4.5 Element update does not affect policy

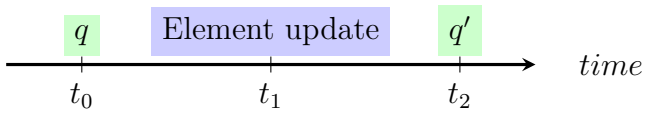

Figure C15: Element update does not affect policy

Figure C15 shows the case when an element is updated between quotes, but this update does not affect the measurements evaluated by the policy. An update does not necessarily mean a change in measurements for all the policies in the policy set of an element. We define this rule, so the rule system knows that if there is an update between quotes and the attested value does not change, this may be acceptable behaviour. Figure C16 shows the logic for the rule.

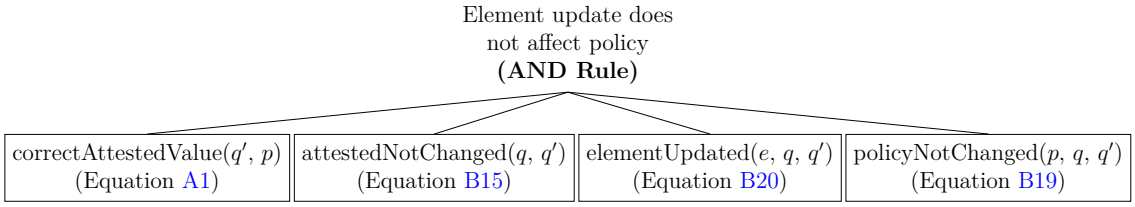

Figure C16: Element update does not affect policy (AND rule)

C.4.6 Detailed checks

Finally, Figure C17 shows a rule that combines the previously introduced rules. This compound rule can be run against an element, and it will be satisfied if the element is in one of the five correct situations described above.

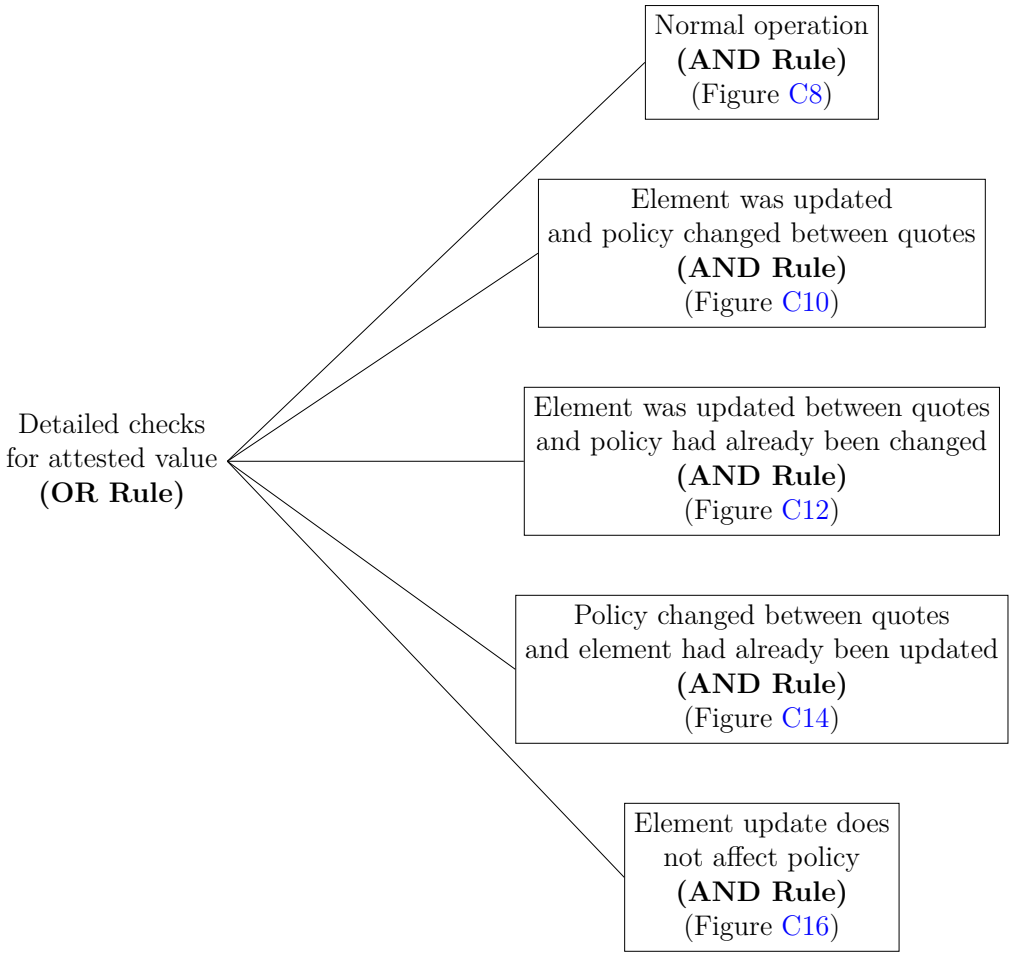

Figure C17: Detailed checks for attested value (OR rule)

D Causal Factor Trees for Root Cause Analysis

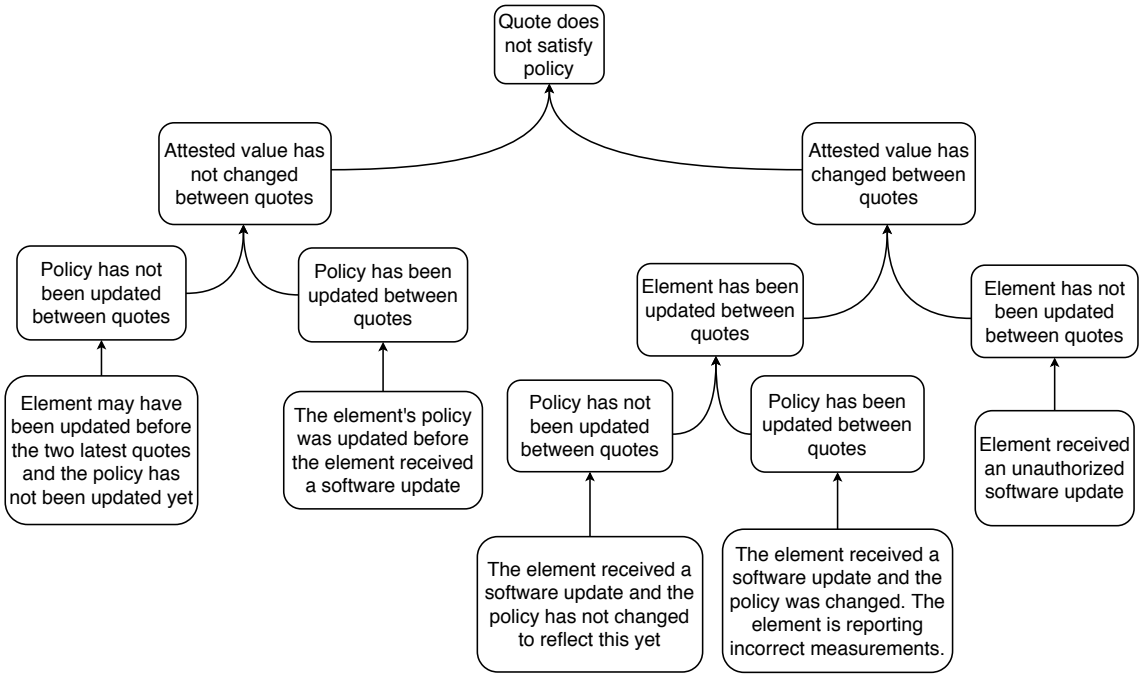

Figure D1: Causal Factor Tree for when a quote does not satisfy the policy

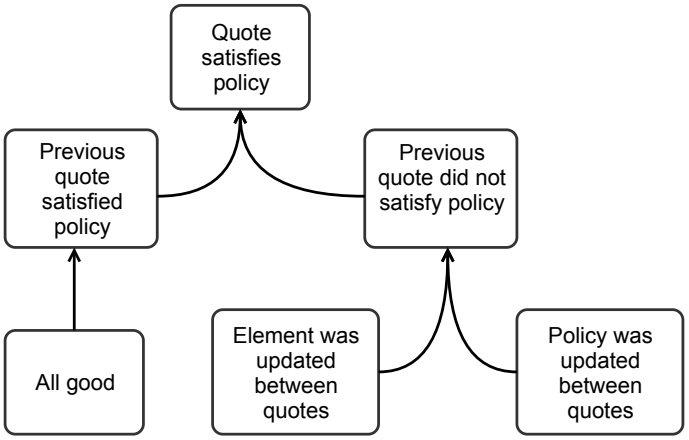

Figure D2: Causal Factor Tree for when a quote satisfies the policy

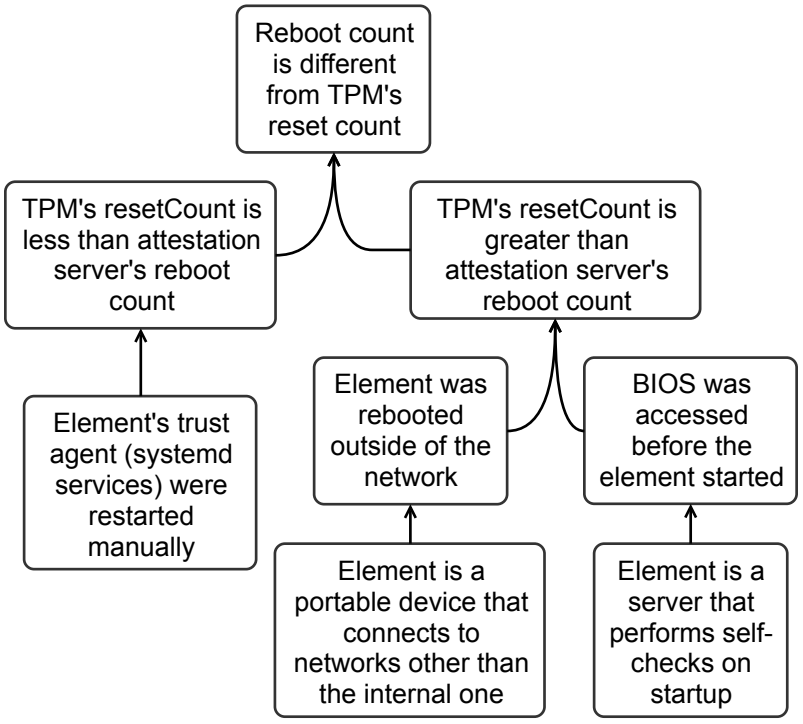

Figure D3: Causal Factor Tree for Reboot trust failure

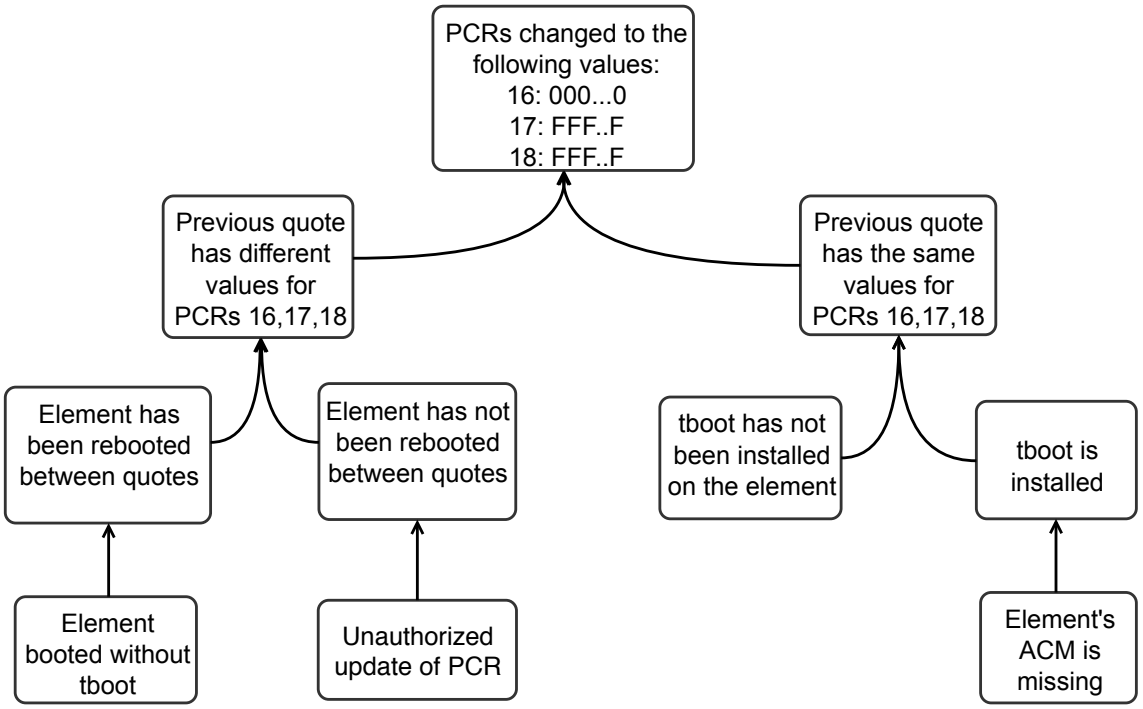

Figure D4: Causal Factor Tree for a missing DRTM measurement

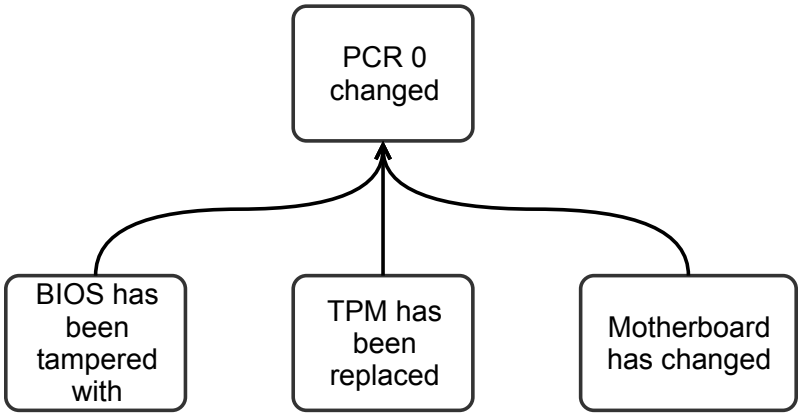

Figure D5: Causal Factor Tree for a change in PCR 0

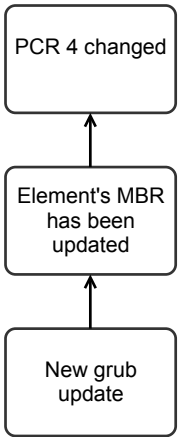

Figure D6: Causal Factor Tree for a change in PCR 4

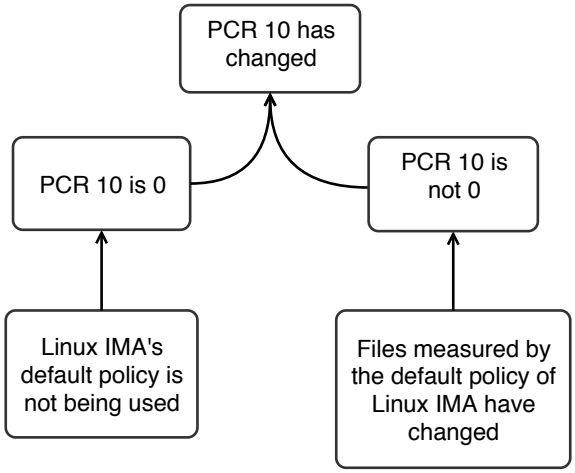

Figure D7: Causal Factor Tree for a change in PCR 10

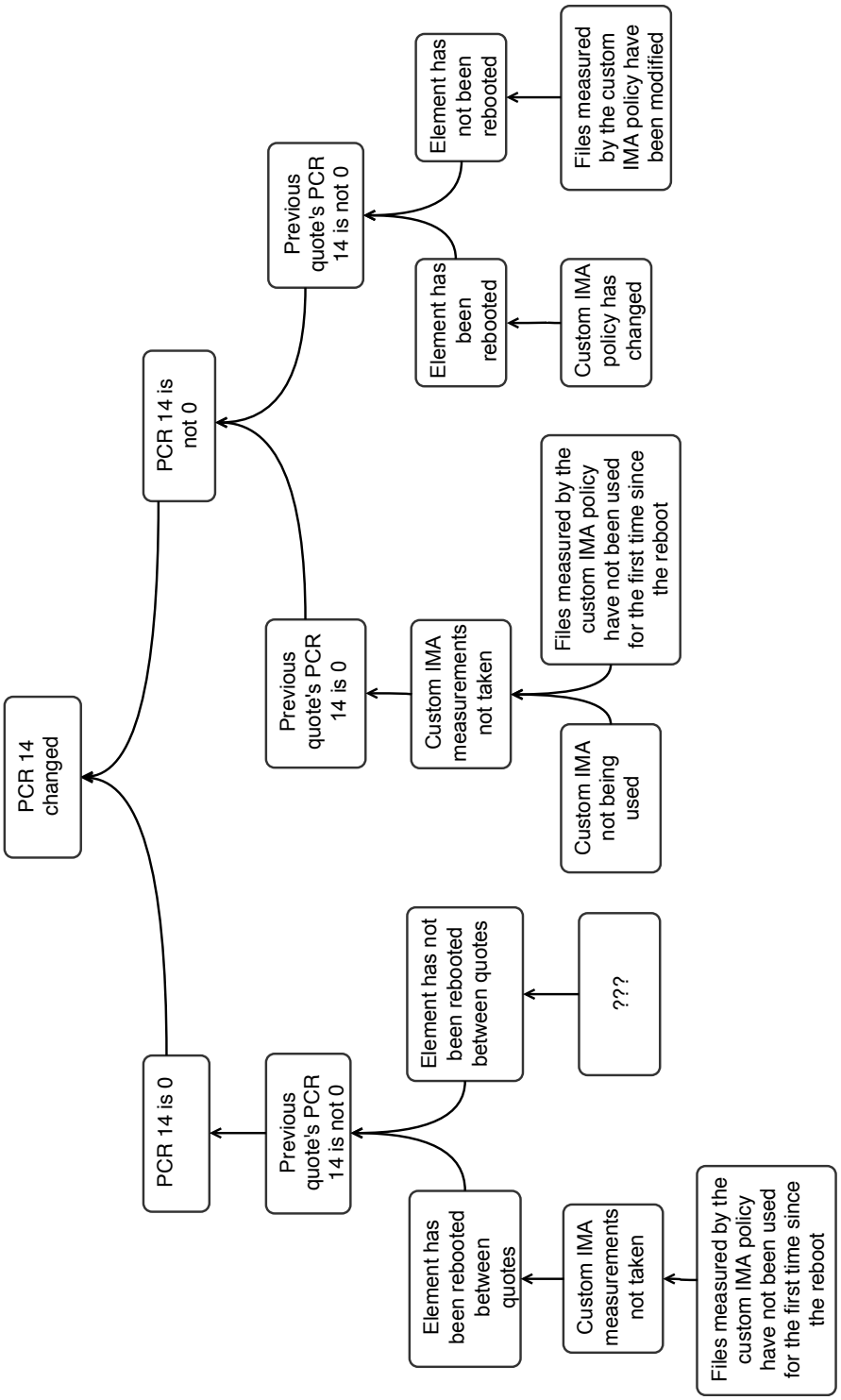

Figure D8: Causal Factor Tree for a change in PCR 14

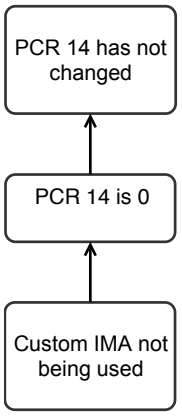

Figure D9: Causal Factor Tree for when PCR 14 shows no change

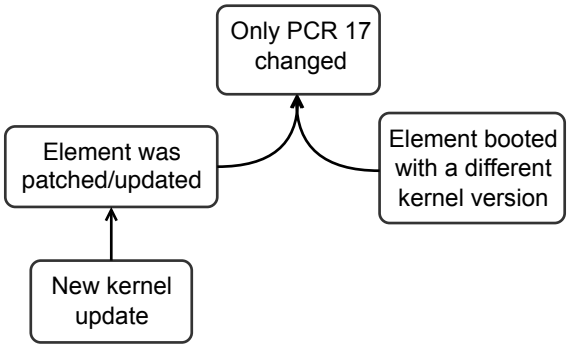

Figure D10: Causal Factor Tree for a change in PCR 17

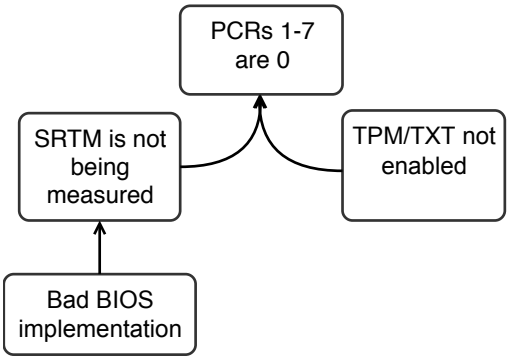

Figure D11: Causal Factor Tree for when PCRs 1-7 are 0

E Ishikawa Diagrams for Root Cause Analysis

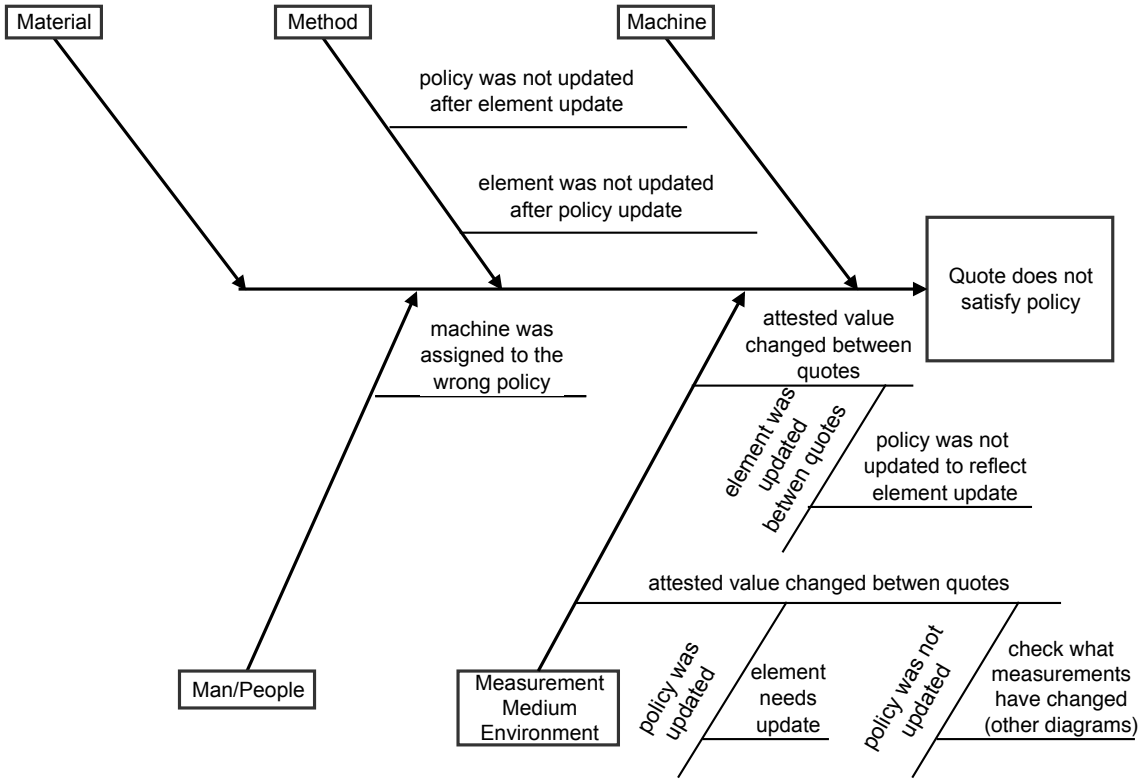

Figure E1: Fishbone diagram for when a quote does not satisfy the policy

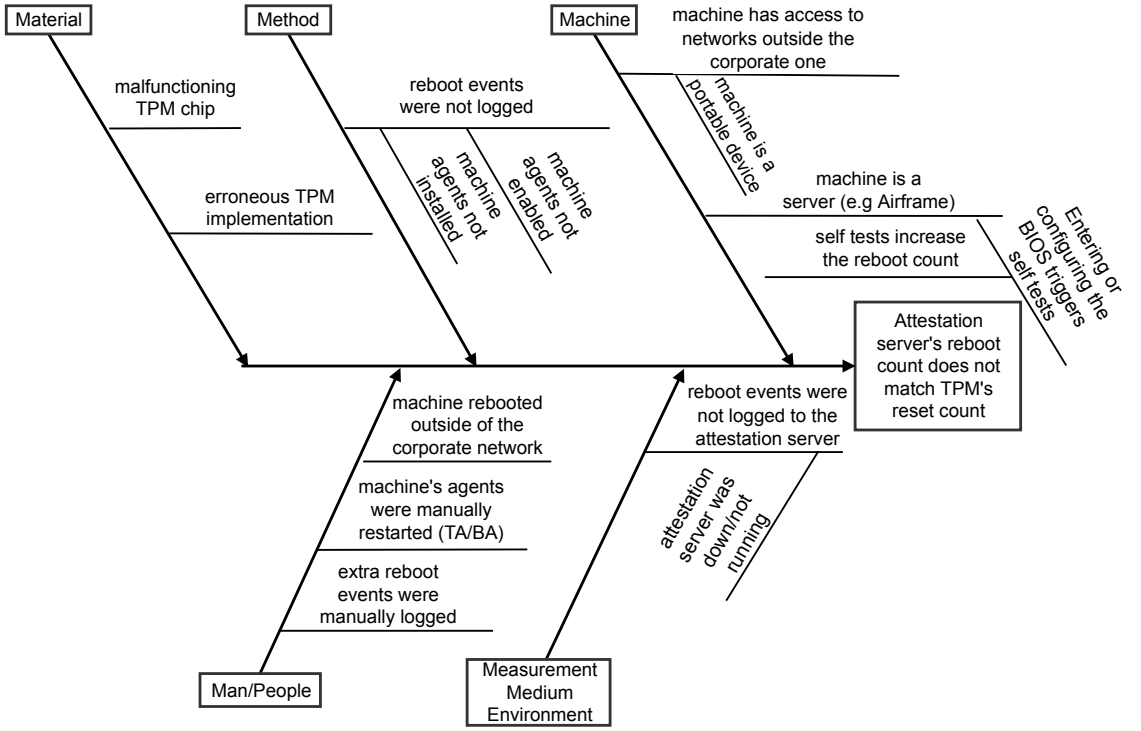

Figure E2: Fishbone diagram for reboot trust failure

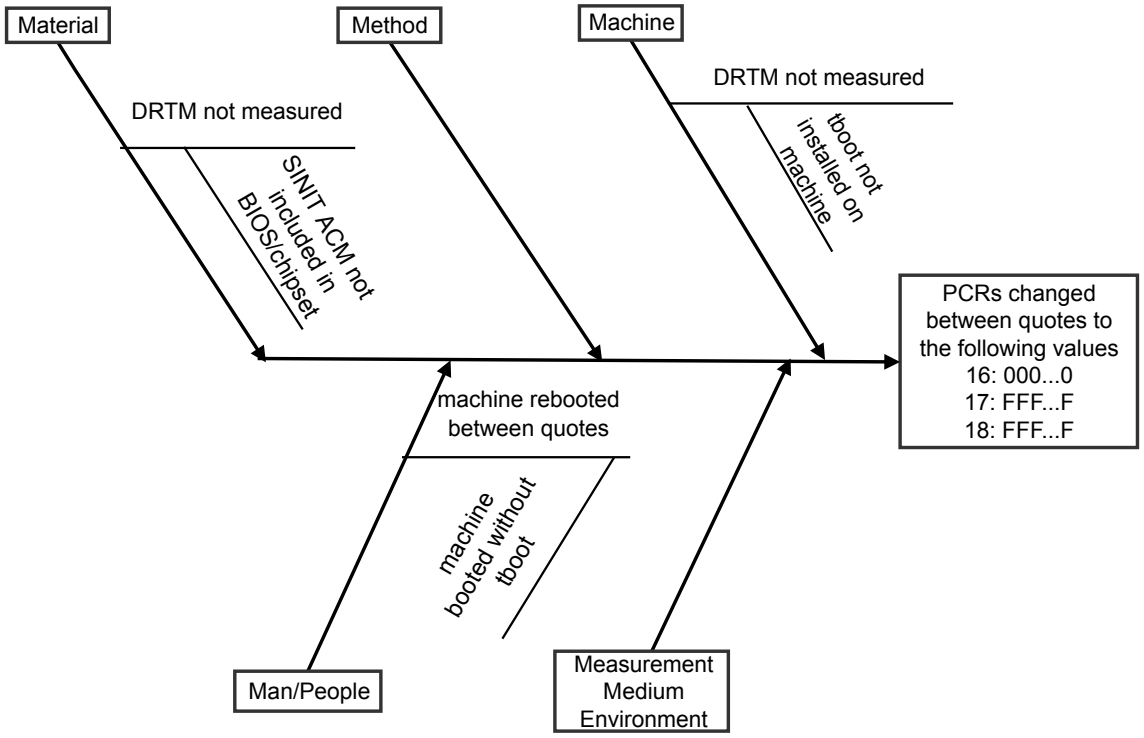

Figure E3: Fishbone diagram for a missing DRTM measurement

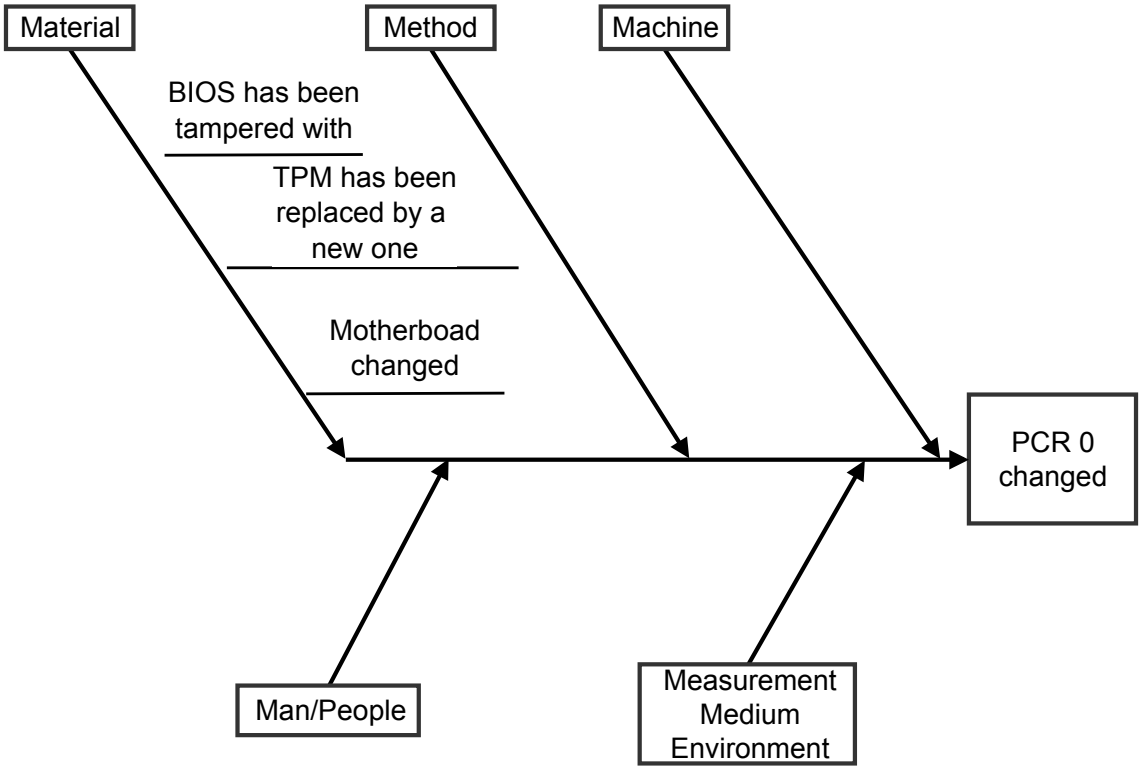

Figure E4: Fishbone diagram for a change in PCR 0

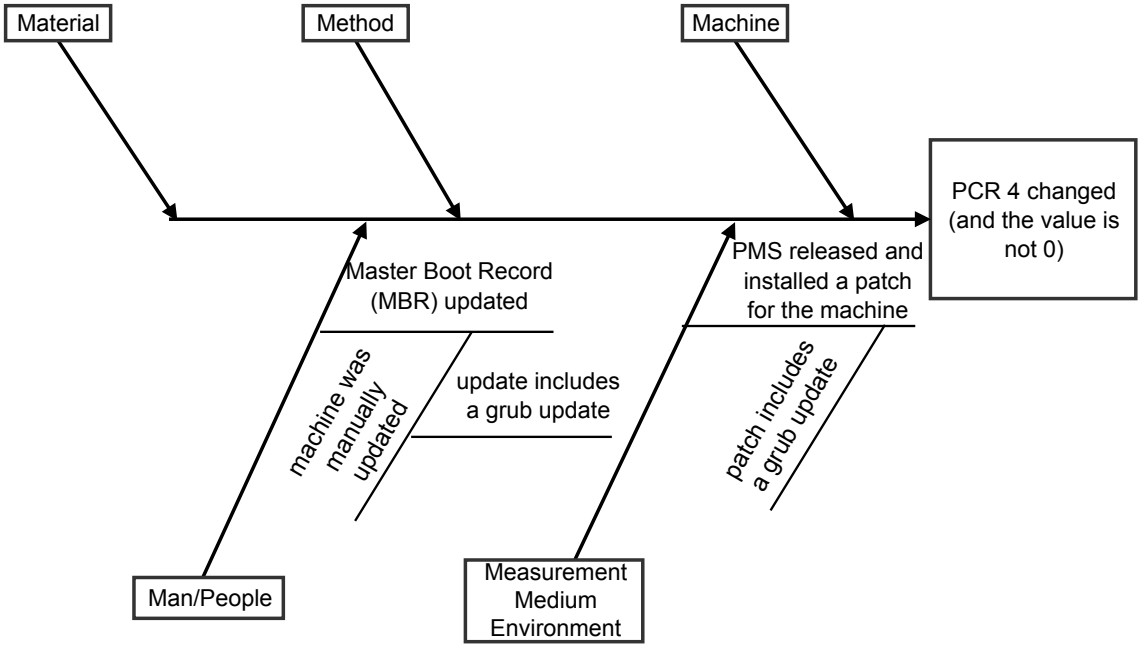

Figure E5: Fishbone diagram for a change in PCR 4

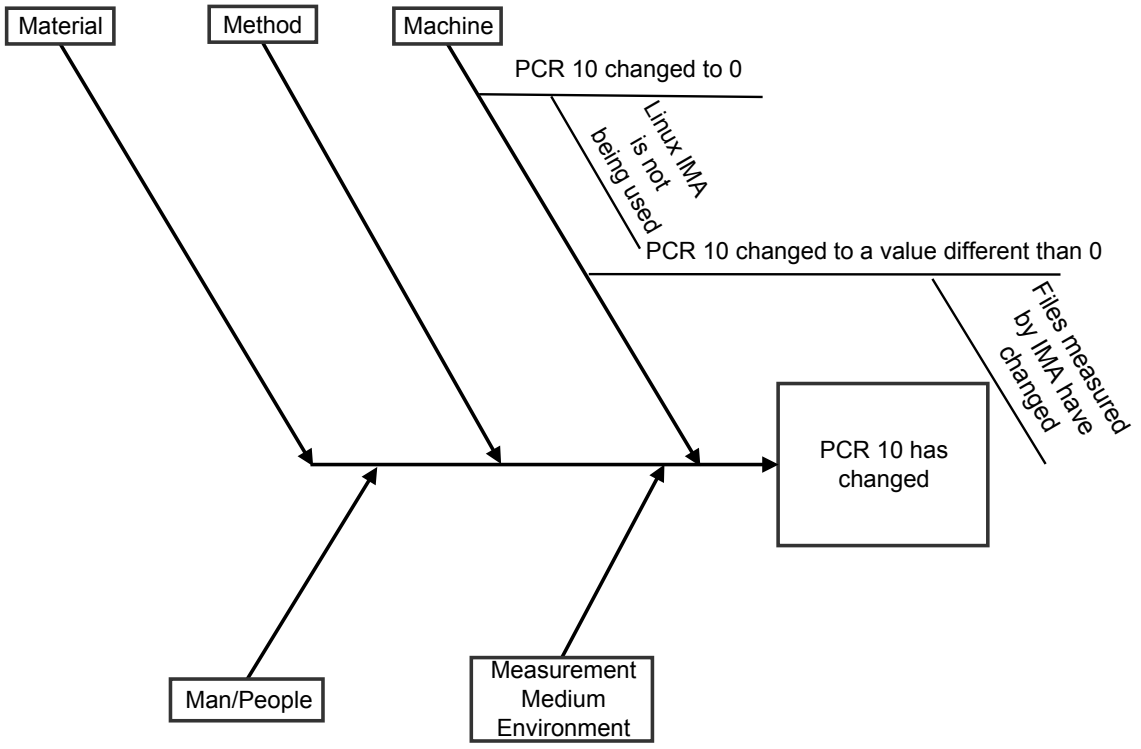

Figure E6: Fishbone diagram for a change in PCR 10

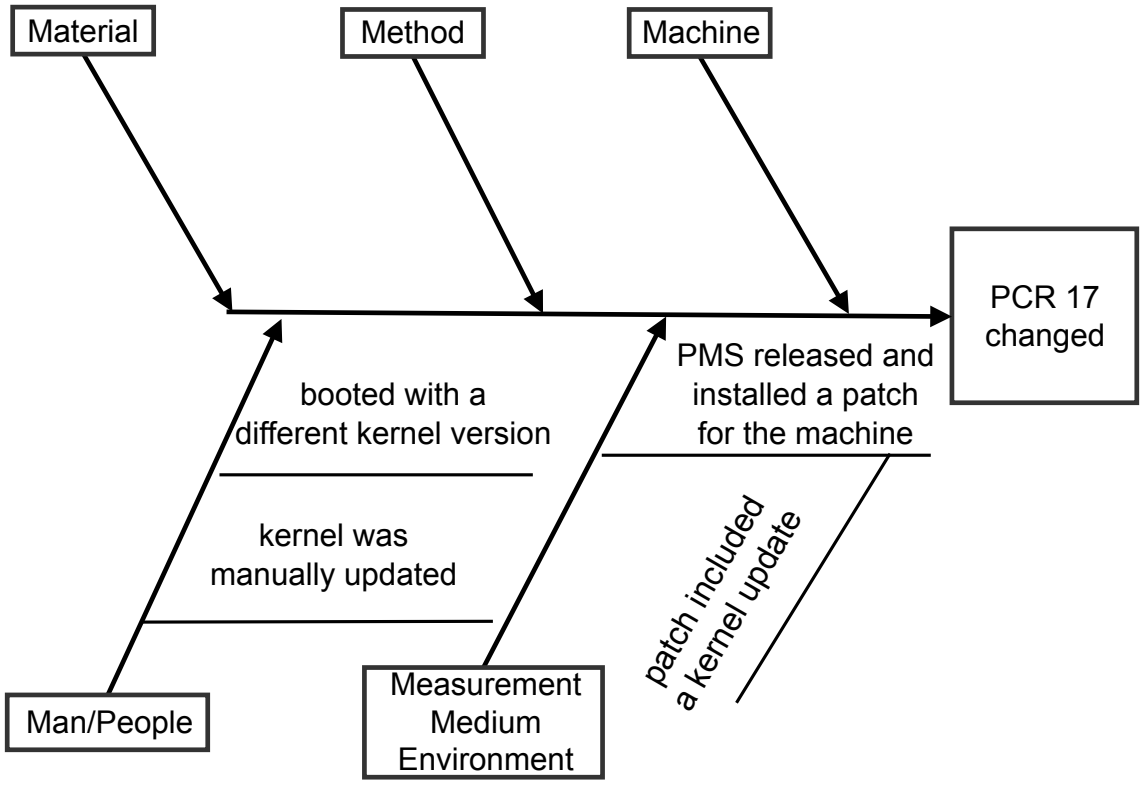

Figure E7: Fishbone diagram for a change in PCR 17

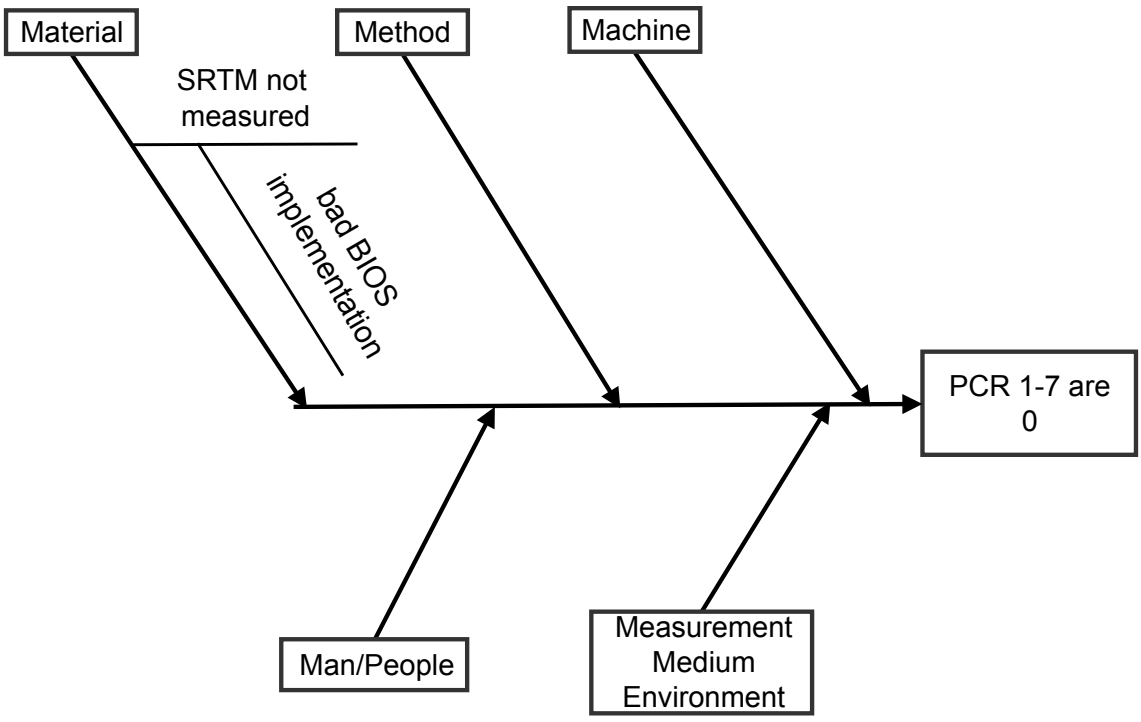

Figure E8: Fishbone diagram for when PCRs 1-7 are 0

F Provisioning of Elements

We define a provisioning module in the attestation libraries, which includes the provisioning functionality. Listing 2 shows the relevant code from that module.

```
"""
.. module:: provisioning
    :synopsis: Module used to provision the NVRAM of an element.

.. moduleauthor:: Gabriela Limonta <gabriela.limonta@nokia.com>
"""
import tempfile
import shutil
import uuid
from tpm2python.libraries import release_nvram, create_policy_for_sealing,
    define_nvram, write_file_to_nvram, read_nvram
from pkg_resources import Requirement, resource_filename


default_config_filename = resource_filename(Requirement.parse("
    attestation_libraries"), "attestation_libraries/example_ta_config.cfg")


def provision_machine_ta_config(config_filename=default_config_filename,
    pcr_policy="sha256:0",
                                nvram_handle="0x1800005", hierarchy="0x40000001",
                                    size=100):
    try:
        tmpdirname = tempfile.mkdtemp()
        policy = "{0}/{1}".format(tmpdirname, uuid.uuid4())
        release_nvram(nvram_handle, hierarchy)
        create_policy_for_sealing(pcr_policy, policy)
        define_nvram(nvram_handle, hierarchy, size, "policyread|policywrite",
            policy)
        write_file_to_nvram(nvram_handle, nvram_handle, config_filename,
            pcr_policy)
        configuration = read_nvram(nvram_handle, nvram_handle, pcr_policy)
        print("Machine successfully provisioned.")
        print("Configuration:")
        print(configuration)
        print("NVRAM index: {0}".format(nvram_handle))
        print("Authorization hierarchy: {0}".format(hierarchy))
        print("NVRAM sealed against PCRS: {0}".format(pcr_policy))
    except Exception as e:
        raise Exception("Error provisioning machine: %s" % str(e))
    finally:
        shutil.rmtree(tmpdirname, ignore_errors=True)


def write_file(filename, contents):
    with open(filename, "w") as file:
        file.write(contents)

```

```
def provision_element_id(element_id, pcr_policy="sha256:0", nvram_handle="0
    x1800006", hierarchy="0x40000001", size=24):
    try:
        tmpdirname = tempfile.mkdtemp()
        policy = "{0}/{1}".format(tmpdirname, uuid.uuid4())
        element_id_file = "{0}/{1}".format(tmpdirname, uuid.uuid4())
        release_nvram(nvram_handle, hierarchy)
        create_policy_for_sealing(pcr_policy, policy)
        write_file(element_id_file, element_id)
        define_nvram(nvram_handle, hierarchy, size, "policyread|policywrite",
            policy)
        write_file_to_nvram(nvram_handle, nvram_handle, element_id_file,
            pcr_policy)
        provisioned_element_id = read_nvram(nvram_handle, nvram_handle, pcr_policy
            )
        print("Machine successfully provisioned.")
        print("Element ID: {0}".format(provisioned_element_id))
        print("NVRAM index: {0}".format(nvram_handle))
        print("Authorization hierarchy: {0}".format(hierarchy))
        print("NVRAM sealed against PCRS: {0}".format(pcr_policy))
    except Exception as e:
        print(e)
        raise Exception("Error provisioning machine: %s" % str(e))
    finally:
        shutil.rmtree(tmpdirname, ignore_errors=True)
```

Listing 2: Provisioning module of the attestation libraries

Then, we define two scripts for the user to run. These scripts are shown in listings 3 and 4 respectively.

```
#!/usr/bin/python

import argparse
from attestation_libraries.provisioning import provision_machine_ta_config
from pkg_resources import Requirement, resource_filename

default_config_filename = resource_filename(Requirement.parse("
    attestation_libraries"), "attestation_libraries/example_ta_config.cfg")

parser = argparse.ArgumentParser(description="Provisions the configuration for a
    machine's trust agent")
parser.add_argument('-c', '--config', help='Path to the configuration for the
    trust agent', dest='config_file',
                    default=default_config_filename, required=False)
parser.add_argument('-s', '--size', help='Size of the NVRAM area', dest='size',
    default=100, required=False)
parser.add_argument('-p', '--pcrs', help='PCRs against which to seal the
    configuration', default="sha256:0",
                    dest='pcrs', required=False)
parser.add_argument('-n', '--nvram-handle', help='NVRAM index', dest='
    nvram_handle', default="0x1800005",
                    required=False)
parser.add_argument('-a', '--auth-hierarchy', help='Authorization hierarchy',
```

```
    dest='hierarchy', default="0x40000001",
                    required=False)


args = parser.parse_args()
provision_machine_ta_config(args.config_file, args.pcrs, args.nvram_handle, args.
    hierarchy, args.size)
```

Listing 3: Trust agent configuration provisioning

```
#!/usr/bin/python

import argparse
from attestation_libraries.provisioning import provision_element_id


parser = argparse.ArgumentParser(description="Provisions the machine with an
    element ID")
parser.add_argument('-e', '--element-id', help='Element ID', dest='element_id',
    required=True)
parser.add_argument('-p', '--pcrs', help='PCRs against which to seal the element
    ID', default="sha256:0", dest='pcrs',
                    required=False)
parser.add_argument('-n', '--nvram-handle', help='NVRAM index', dest='
    nvram_handle', default="0x1800006",
                    required=False)
parser.add_argument('-a', '--auth-hierarchy', help='Authorization hierarchy',
    dest='hierarchy', default="0x40000001",
                    required=False)

args = parser.parse_args()
provision_element_id(args.element_id, args.pcrs, args.nvram_handle, args.
    hierarchy)
```

Listing 4: Element ID provisioning